POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Executive Summary of the Thesis

Constrained path following and dynamic occlusion avoidance for floating base robotic systems

Laurea Magistrale in Automation and Control Engineering Engineering - Ingegneria dell'Automazione

Author: Ibrahim Ibrahim

Advisor: Prof. Paolo Rocco

Co-advisor: Prof. Marco Hutter

Academic year: 2020-2021

1. Introduction

The goal of this thesis is two-fold. The first is more application-oriented and aims to plan a motion that follows a path under constraints. The main idea is that the robot needs to follow a geometric path with no assigned speed optimally. This is a widely studied problem and is most commonly applied to robot manipulators in literature. Basically, since the timing of the path is unspecified, there is an additional degree of freedom in time which may be exploited directly within motion planning to control the rate of progression along the path.
The second goal is more research oriented and it aims to achieve whole-body motion planning of the robot such that shadows cast by obstacles occluding the line of sight between the end-effector and the target are avoided. More specifically, the problem setup is one where the robot's end-effector is supposed to interact with a target of interest in an environment, be it manipulation, tracking, detection, or any other task. The environment is cluttered with objects that occlude the line of sight between the end-effector and the target prohibiting as a result the completion of any meaningful task. If the target is considered to be a light source, one may require the end-effector to avoid the shadows cast by the occluding obstacles, and to remain instead in positions where there exists a line of sight.
One typical way of achieving the second goal in optimal control literature is to use visibility constraints. Visibility constraints, also known as two-dimensional constraints, are often used in Model Predictive Control (MPC) schemes to keep the image-plane coordinates of interest within the camera's Field of View (FOV) [2]. Such constraints can also represent forbidden regions in the image. Using this type of constraint has some limits. For example, control, in this case, is typically interrupted if the target goes out of FOV. Mezouar and Chaumette [6] use a softened visibility constraint which is reformulated as a repulsive potential field, but such a constraint highly increases the chance of having local minima in the overall potential field. This type of solutions is also limited to working in a visual servoing framework.
Otherwise, one would have to resort to methods which tackle occlusion avoidance on a higher level separately than motion planning itself. Those methods include active vision techniques.

Active vision methods try to optimize the positioning of the sensor so as to maximize the amount of useful sensory data. Examples of those methods are next-best view (NBV) planning techniques which select the next pose based on some criteria and then pass the pose to the motion planner. Most NBV planning methods are still immature in terms of efficiency and efficacy and are nowhere near real-time. Wu et al. [8] perform NBV planning by evaluating the visibility as well as the likelihood of feature matching, achieving around 1.5s per step. On the other hand, [5] plans the NBV by utilizing online aspect graphs to account for occlusions and feature visibility, requiring more than a minute of planning per movement.

2. Problem Formulation

The second goal is to achieve a holistic motion planning formulation for whole-body control of the robot while maximizing the probability of target visibility. Due to the multi-objective nature of the problem at hand, it is suitable to adopt a finite horizon optimal control formulation, which is solved in an MPC fashion. The optimal control problem is formulated in the continuous-time domain as

$$\begin{aligned}
&\underset{\boldsymbol{u}(\cdot)}{\text{minimize}} && \phi(\boldsymbol{x}(t_f)) + \int_{t_s}^{t_f} l(\boldsymbol{x}(t), \boldsymbol{u}(t))dt && \text{(1a)}\\
&\text{subject to:} && \boldsymbol{x}(t_s) = \boldsymbol{x}_s && \text{(1b)}\\
& && \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) && \text{(1c)}\\
& && \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t) = \boldsymbol{0} && \text{(1d)}\\
& && \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, t) \geq \boldsymbol{0}, && \text{(1e)}
\end{aligned}$$

where $\boldsymbol{x}$, $\boldsymbol{u}$ are the state and control input respectively. MPC finds the minimum-cost (1a) trajectory under the system dynamics $\boldsymbol{f}$, the equality constraints $\boldsymbol{g}$, and the inequality constraints $\boldsymbol{h}$. In practice, $\boldsymbol{u}(t_s)$ is then applied to control the robot in the receding horizon fashion to allow re-planning with a new measured state, $\boldsymbol{x}_s$. The problem is solved using Sequential Linear-Quadratic (SLQ) solver [3] provided by the OCS2 toolbox [1]. The visibility and orientation costs discussed ahead are considered to be acting on the robot end-effector, but they can generally act on any other frame. It is assumed that the camera that drives potential object detection and tracking is mounted on the end-effector. The first goal may also be derived using the same formulation, obtaining as a result constrained SLQ MPC path following.

2.1. Path Following SLQ MPC

This formulation is specific to a 7DOF mobile manipulator. The path to be followed by the end-effector is arc-length parametrized, with θ being the arc-length parameter. The path starts at $\theta = 0$ and ends in $\theta = 1$. This parameter is augmented within the system state vector, x, forming the augmented system state vector

$$\boldsymbol{x}_a = \begin{bmatrix} b_x & b_y & b_\psi & q_1 & \dots & q_7 & \theta \end{bmatrix}, \tag{2}$$

having a size $n_{x_a} = 3 + n_j + 1 = 11$, where b_x, b_y, and b_{psi} are the x, y, and yaw degrees of freedom of the mobile manipulator base (differential drive platform) and $n_j = 7$ refers to the degrees of freedom of the manipulator.
Setting the dynamics of our arclength parameter as

$$\dot{\theta} = \omega(t), \tag{3}$$

allows us to augment ω in the input vector as

$$\boldsymbol{u} = \begin{bmatrix} v_b & \dot{\psi} & \dot{q}_1 & \dots & \dot{q}_7 & \omega \end{bmatrix} \tag{4}$$

resulting in $n_u = 10$ which gives us full control over the rate of progression of the arclength parameter. As such, the progression along the path can be commanded by constraining it to a desired value, ω_{des}, in a soft input constraint that penalizes the deviation as

$$\int_{t_s}^{t_f} \|\omega_{des} - \omega(t)\|_2^2 \, dt. \tag{5}$$

If ω_{des} is set to 1, one would be asking for a behaviour that resembles the behaviour of a Trajectory Optimization (TO). Otherwise, one can dynamically set a faster, $w_{des} > 1$, a slower, $0 < w_{des} < 1$, or even a negative, $w_{des} < 0$ path progression relative to TO.
The soft input constraint (5) controls the arclength parameter, but the end-effector still needs to follow the path at the emerging arclength value. That can be done by adding a soft state cost that penalizes the difference between the current end-effector pose and the desired end-effector pose, the latter being specified by the arclength parametrization. At every step, a θ emerges based on the (3) and the pose along the path at the emerging θ will be the desired

one. The current pose is directly extracted from the fed back measurements. That is, the error below is penalized

$$\boldsymbol{x}(t) - \boldsymbol{x}_{des}(\theta). \tag{6}$$

2.2. Whole-Body MPC and Dynamic Occlusion Avoidance

As mentioned earlier, visibility constraints in control problems are typically enforced by restricting the pixel coordinates of the tracked/detected object of interest within upper and lower bounds inside the current FOV as shown in (7)

$$\begin{bmatrix} u_{min} \\ v_{min} \end{bmatrix} \leq \begin{bmatrix} u_{act} \\ v_{act} \end{bmatrix} \leq \begin{bmatrix} u_{max} \\ v_{max} \end{bmatrix}. \tag{7}$$

We reformulate those constraints as a maximization of the likelihood that the end-effector has a line of sight to the target. The likelihood function, however, is not time-separable, making it unsuitable for trajectory optimization methods. As such, rather than maximizing the likelihood of visibility, we maximize the log-likelihood

$$\int_{t_s}^{t_f} \log \mathcal{F}(\boldsymbol{x}_{ee}(t))dt, \tag{8}$$

where $\mathcal{F}$ in (8) is the *shadow field* containing information about visibility and occlusion and $\boldsymbol{x}_{ee}$ is the 3D position of the end-effector frame of interest within that field. Since we formulated our problem as a minimization, we may augment (8) within (1a) as

$$l_{aug}(\boldsymbol{x}, \boldsymbol{u}) = l(\boldsymbol{x}, \boldsymbol{u}) - \log \mathcal{F}(\boldsymbol{x}_{ee}), \tag{9}$$

where the first term in (9) represents motion planning and system costs, and the second term represents the visibility cost. The gradient of such a cost can be obtained since the gradient of $\boldsymbol{x}_{ee}$ is the Jacobian of the end-effector, and the probabilistic *shadow field* is continuous and smooth, allowing for efficient on-demand trilinear gradient computation at every step. Since the values returned by $\mathcal{F}(\boldsymbol{x}_{ee})$ can be zero, (8) is susceptible to extreme values due to the log function, making it unsuitable for shooting methods. Therefore, we use the relaxed log barrier penalty introduced in [4].
The proposed reformulated visibility constraint acts only on the end-effector position, leaving the end-effector orientation unconstrained. As a soft constraint, we add a cost that locks the end-effector onto the target, ensuring that the target remains within the camera's FOV, further enhancing the tracker's confidence about the target. Such a constraint is a function of the emerging solution from the planner along every step of the horizon. By properly scaling this constraint's penalty, we safeguard against sudden tracker updates and faults. In this manner, the end-effector gradually and progressively locks onto the target. We add the corresponding cost to l_{aug}, constituting the second term of the resulting total cost function as

$$l_{total}(\boldsymbol{x}, \boldsymbol{u}) = l_{aug}(\boldsymbol{x}, \boldsymbol{u}) + l_o(\boldsymbol{x_{ee}}). \tag{10}$$

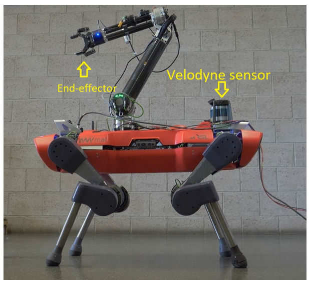


Figure 1: Photo showing ALMA, a robot composed of a custom-made, torque-controllable 4 degrees of freedom robotic arm, DynaArm, mounted on a quadrupedal platform, ANYmal C. We highlight the position of the Velodyne sensor and the robot end-effector in ALMA's standing position.

3. *Shadow Field*

The likelihood of visibility represented by $\mathcal{F}$ is obscure and not readily available, so we propose a novel efficient dynamic-programming algorithm called *shadow field* that returns a field containing visibility likelihood values at every position in our scene. Such a field is constructed using LIDAR or RGB-D data and provides soft shadow approximations of hard shadows.

4. Results

In this section, simulation results validating the SLQ MPC path following formulation are presented for the case of a mobile 7DOF manipulator, namely Kinova 7DOF mounted on a differential drive platform. Moreover, the proposed *shadow field* algorithm is validated on real data from hardware and the performance of the complete whole-body MPC and dynamic occlusion avoidance control pipeline is demonstrated in simulation on ALMA (Fig. 1). ALMA is equipped with four Realsense D435 cameras that provide 360° vision as well as a VLP16 Puck LITE Velodyne sensor mounted on the robot's rear. It has two on-board computers, one running locomotion and planning modules, and the other running mapping modules. The adopted planning framework in our physics-based experiments is based on the whole-body motion planner formulation introduced in [7]. We then augment our visibility and orientation costs as discussed in Section 2.

4.1. Path Following SLQ MPC

The result for the case of collision avoidance is presented. Further case studies are available in the main manuscript. In this scenario, we place an obstacle along the path. We can detect this obstacle's location and infer if there will be a collision with it or not along the path, and if yes, infer the value of θ at which there will be collision. We show the evolution of the behaviour of the robot in Fig. 2 where the robot pauses its progression right before collision, i.e., achieving collision avoidance. In Fig. 2, the progression is visibly killed. The solver tries to proceed at different times along the path but since progression is unfavorable, the robot's end-effector successfully avoids collision.

4.2. Shadow Field Mapping

Two real-time mapping experiments are considered. In each of them, ALMA is placed in a different cluttered environment, and data from the Veldoyne sensor are used to construct a 3D probabilistic occupancy grid. The occupancy grid is then used to compute and publish a 3D *shadow field* spanning $16 \times 16 \times 2m^3$ at a resolution of 1000 voxels per cubed meter. The onboard mapping computer runs the whole pipeline at rates exceeding 100Hz, far beyond the Velodyne pointcloud update rate (~15Hz). Since it is not possible to visualize the whole 3D *shadow field* in a meaningful manner, we visualize only a horizontal slice of it at a defined height in the form of a pointcloud having gray-scale intensities proportional to the values of the *shadow field*. At that slice level, brighter areas represent visible regions while darker areas represent occluded regions. We publish this slice at the robot's end-effector level. In both experiments, we coincide the light position with the end-effector position at the center of the field, i.e., we are evaluating the end-effector's sight of the surrounding scene. We also visualize the occupied voxels,

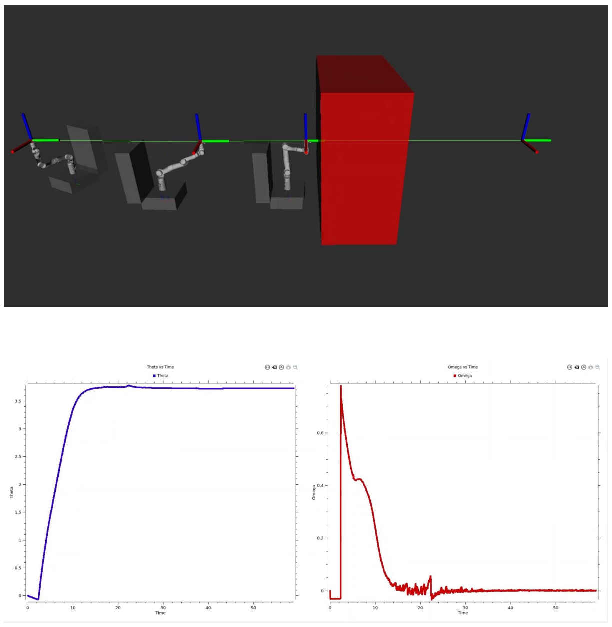


Figure 2: Multiplicity illustration showing the evolution of the robot end-effector along the path throughout a nominal path following setting with collision avoidance. Plots of arclength θ in blue and ω in red for the case of collision avoidance. Progression freezes indefinitely right before collision.

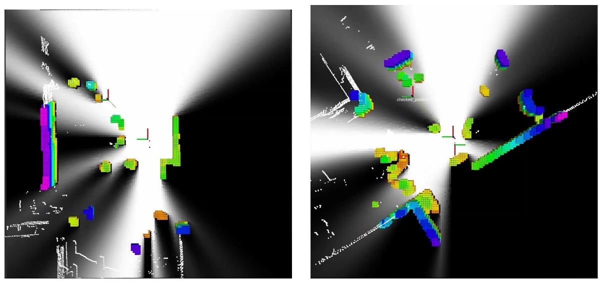


Figure 3: Top view of the occupied voxels (colored), the Velodyne pointcloud (small white dots), and the resulting *shadow field* pointcloud slice (gray-scale). The light source and the end-effector are at the center of each image. The umbras produced by the occluding voxels are dark while their penumbras are the shades of gray.

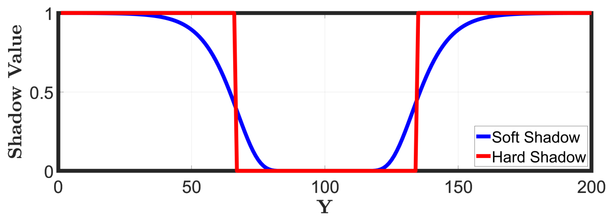


Figure 4: Values of our soft *shadow field* approximation against hard shadow values produced by an obstacle.

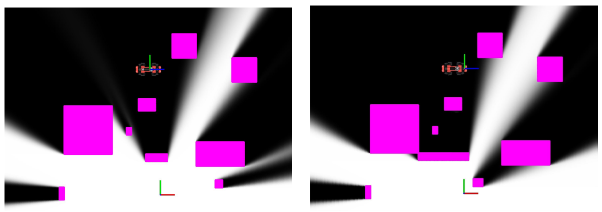


Figure 5: Simulation scenes containing obstacles shown in violet, ALMA shown as the top frame, and the light source shown as the bottom frame.

which have different colors correlated to their occupation probability. We show the top view of the slice, the occupied voxels, and the Velodyne pointcloud in Fig. 3. Asides from noise arising at the stage of building the occupancy grid, the resulting shadows retain the same smoothness and continuity characteristics as the soft shadow presented in Fig. 4.

4.3. Motion Planning and Control Pipeline

Two experiments running in a physics-based simulation environment, Gazebo, that validate our motion planning and control pipeline are presented. The MPC is solved at a 1.0s horizon, and the relevant penalties arise from the actuator inputs, the visibility and orientation costs, height tracking, and the actuator speed, position, and torque limits. Other regularization costs and gait-related constraints are active during this process. We also provide the simulation with a given occupancy grid built based on simulated obstacles in the environment. The grid resolution we adopt is 1000 voxels per cubed meter. In Fig. 5, we illustrate the top view of each of the two scenes in which we carry out the simulation. In both illustrations, a slice of the *shadow field* at the end-effector level is shown in gray-scale with the same intensity scale as

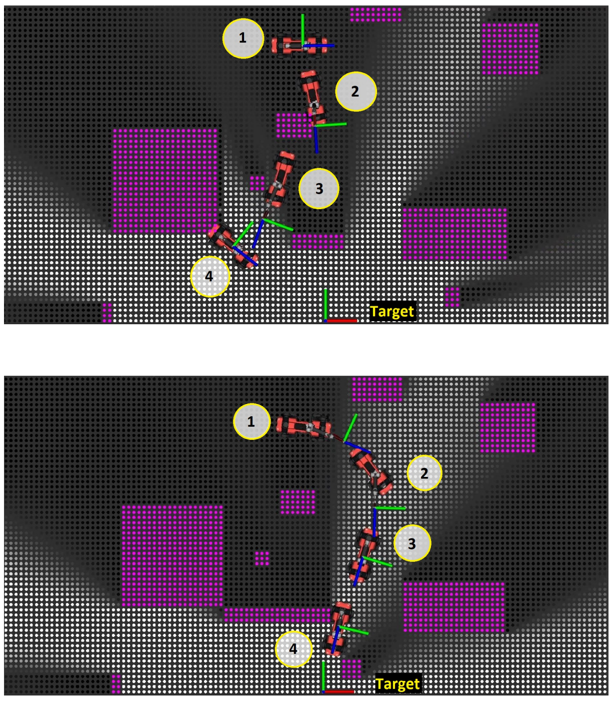


Figure 6: Multiplicity illustrations of our physics-based simulations in the two scenes introduced in Fig. 5. The solver solution in each simulation is reflected through 4 different snapshots of the robot. The target is tagged. Violet corresponds to obstacles at end-effector level.

previously introduced for Fig. 3. Umbras and penumbras described there can also be noted here. As shown in Fig. 5, we make sure that the end-effector starts in an occluded position. The robot must then plan a least-shadowy path for the end-effector position that leads it to establish line-of-sight with the light position.
In Fig. 6, we illustrate the robot's motion and its end-effector frame pose in a top view. In the first simulation, snapshot 1 shows the robot in its starting occluded position. The second snapshot shows the end-effector sliding along the surface of an occluding obstacle so as to circumnavigate it. The third snapshot shows the end-effector being extended ahead of the robot, trying to reach locations with higher visibility. The last snapshot shows the robot in a steady state. Having reached full visibility, the end-effector is free to lock onto the target. We achieve an orientation error complement of 0.995. The second scene is similar to the first one, except that we block the path of the previous solution by extending the first obstacle facing the light frame. We also

move one of the obstacles and place it right next to the light position. The robot directly extends the end-effector into the visible region in snapshot 1, then traverses the slightly shadowed region and peaks its end-effector through the tight passage by the fourth snapshot to achieve full visibility. As an interesting emergent behavior, the end-effector avoids collisions with obstacles since their *shadow field* values are zero. This behavior may be extended to other robot frames.

5. Conclusions

In this thesis, we formulated a constrained path following SLQ MPC problem for a mobile manipulator. We embedded the arclength variable with which the desired path is parametrized within the system states, giving the solver full control over the rate of progression as it deems fit to accommodate for arising costs and path constraints. We implemented this formulation for a 7DOF manipulator and we studied its behaviour for three cases. For the nominal case, we provided a qualitative and quantitative description of the manipulator's path following performance. For the case of a wrong initialization, we highlighted the solver's ability to correct the initial error by backtracking the solution of the arclength and proceeding nominally once the end-effector is back on track. For the last case, we show the solver's ability to avoid collisions by placing a relaxed-log barrier at the arclength value where a collision is expected to happen.
In this work, we also proposed an MPC formulation based on visibility constraints. We augmented our motion planning cost function with a penalty maximizing relaxed log-likelihood of visibility probability. We introduced a probabilistic *shadow field* that quantifies visibility probability based on the occupancy map of the scene. We validated the quality of this map in simulation and hardware. We further discussed the computational and storage complexities of our *shadow field* mapping and showcased its computational efficiency for onboard applications by real-time mapping on ALMA hardware. A comparison between hard and soft shadows for one 2D implementation shows the extent and accuracy of our approximation. Last but not least, we demonstrated the validity of our proposed MPC formulation for motion planning and dynamic occlusion avoidance in simulation for ALMA.

6. Acknowledgements

This thesis would not have been possible without the support and mentorship of my supervisors at Robotic Systems Lab at ETH Zurich: Dhionis Sako, Perry Franklin, Farbod Farshidian, and Jan Preisig.

References

[1] OCS2: An open source library for optimal control of switched systems. [Online]. Available: `https://github.com/leggedrobotics/ocs2`.

[2] Guillaume Allibert, Estelle Courtial, and François Chaumette. Predictive control for constrained image-based visual servoing. *IEEE Transactions on Robotics*, 26(5):933–939, 2010.

[3] Farbod Farshidian, Michael Neunert, Alexander W Winkler, Gonzalo Rey, and Jonas Buchli. An efficient optimal planning and control framework for quadrupedal locomotion. In *ICRA*, pages 93–100. IEEE, 2017.

[4] Ruben Grandia, Farbod Farshidian, René Ranftl, and Marco Hutter. Feedback mpc for torque-controlled legged robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4730–4737. IEEE, 2019.

[5] C. McGreavy, Lars Kunze, and Nick Hawes. Next best view planning for object recognition in mobile robotics. In *PlanSIG*, 2016.

[6] Y. Mezouar and F. Chaumette. Path planning for robust image-based control. *IEEE Transactions on Robotics and Automation*, 18(4):534–549, 2002.

[7] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti, and Marco Hutter. A unified mpc framework for whole-body dynamic locomotion and manipulation, 2021.

[8] Kanzhi Wu, Ravindra Ranasinghe, and Gamini Dissanayake. Active recognition and pose estimation of household objects in clutter. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4230–4237, 2015.

POLITECNICO
MILANO 1863

School of Industrial and Information Engineering
Master of Science in Automation and Control Engineering

Master Thesis

Constrained path following and dynamic occlusion avoidance for floating base robotic systems

Academic year 2020-2021

Supervised by:

Prof. Paolo Rocco

Prof. Marco Hutter

Author:

Ibrahim Ibrahim

Student ID: 940160

Contents

Contents **ii**

Preface **iii**

Abstract **v**

Sommario **vii**

Notations **ix**

List of Figures **xi**

List of Tables **xiii**

List of Algorithms **xv**

1 Introduction **1**
- 1.1 Outline and Contribution . . . 2
- 1.2 Literature Review . . . 4

2 Preliminaries and System Overview **11**
- 2.1 Kinematics . . . 11
- 2.2 Dynamics . . . 14
- 2.3 Kinova 7DOF Mobile Manipulator . . . 15
- 2.4 Articulated Locomotion and MAnipulation - ALMA . . . 18
- 2.5 Constrained Numerical Optimization for Continuous-Time OCPs . . 21

3 Problem Formulation **27**
- 3.1 Path Following SLQ MPC . . . 28
- 3.2 Dynamic Occlusion Avoidance within Whole-Body Motion Planning 33
  - 3.2.1 The Reformulated Visibility Constraint . . . 33
  - 3.2.2 The Orientation Constraint . . . 34
  - 3.2.3 Equality and Inequality Constraints . . . 36

4 *Shadow Field* **37**
- 4.1 The Visibility Problem . . . 38
- 4.2 *Shadow Field* Algorithms . . . 38

5 Experimental and Physics-Based Simulation Results **49**
- 5.1 Path Following . . . 50
  - 5.1.1 Path Generation & Numerical Arclength Parametrization . . 50
  - 5.1.2 Nominal case . . . 52
  - 5.1.3 Wrong Initialization Case . . . 53

5.1.4 Collision Avoidance Case 54
5.1.5 Qualitative and Quantitative Discussion for the Nominal Case 55
5.2 Dynamic Occlusion Avoidance Within Whole-Body Motion Planning 57
5.2.1 Motion Planning and Control Pipeline 57
5.2.2 Realtime *Shadow Field* Mapping 58
5.2.3 Physics-based Simulations 60

6 Future Work **63**

7 Conclusion **65**

Bibliography **69**

A Text Appendix **71**
A.1 Fixed Base Kinematics 71
A.2 Fixed Base Dynamics 73
A.3 Estimators 76
A.3.1 Kalman Filter 76
A.3.2 Moving Horizon Estimator 78
A.4 A Fast Visibility Check Using Horizon Culling 79

B Graph Appendix **81**
B.1 Deterministic Hard Shadows and a Gradient-Based Solution 81
B.2 More Kinova and ALMA Simulations 82

C Datasheets **89**

Preface

The main reason I joined the MSc program in Automation and Control Engineering at Politecnico di Milano was my wish to work on designing and implementing control systems for robotic applications. This thesis fulfills the requirement for my MSc degree and marks yet another milestone in my journey through robotics, a field I have been passionate about since high-school.

This work has been carried out at the Robotic Systems Lab (RSL) in the Department of Mechanical and Process Engineering at ETH Zurich as part of the Swiss-European Mobility Programme. The projects I worked on further endow mobile manipulators and legged robots with human-like understanding of path following and occlusion avoidance. Throughout my thesis, I had the mindset of finding the simplest solutions to the problems I faced as I am a strong believer that simpler solutions to engineering problems work best. The main challenges during this thesis were the limited physical access to the robots in the lab and the learning curve of the programming libraries and methods of the researchers at RSL.

I am grateful to professors Paolo Rocco and Marco Hutter for supporting my thesis. I am also grateful to my supervisors at RSL, Dhionis Sako, Perry Franklin, Farbod Farshidian, and Jan Preisig for their generous mentorship. Lastly, I am grateful to my family for their endless love and support. It has been an incredibly rich learning experience!

Abstract

This work formulates a constrained path following sequential-linear quadratic model predictive control (MPC) problem for floating base manipulators. By augmenting the path's arclength parameter within the system states, we can exercise direct control over the path's rate of progression. We exploit this added degree of freedom to regulate progression along a path so as to accommodate for the various dynamic costs and constraints. Through simulations on a Kinova seven degrees of freedom mobile manipulator, we validate our formulation and demonstrate the solver's plasticity in different scenarios. Several settings are studied and presented, namely a nominal case, the case of avoiding collisions, and the case of recovery from wrong initialization. A brief qualitative and quantitative study on the nominal case results is provided.

This work also introduces a novel approach for whole-body motion planning and dynamic occlusion avoidance. The proposed approach reformulates the visibility constraint as a likelihood maximization of visibility probability. In this formulation, we augment the primary cost function of a whole-body MPC scheme through a relaxed log barrier function yielding a relaxed log-likelihood maximization formulation of visibility probability. The visibility probability is computed through a probabilistic *shadow field* that quantifies point light source occlusions. We provide the necessary algorithms to obtain such a field for both 2D and 3D cases. We demonstrate 2D implementations of this field in simulation and 3D implementations through real-time hardware experiments. We show that due to the linear complexity of our *shadow field* algorithm to the map size, we can achieve high update rates, which facilitates onboard execution on mobile platforms with limited computational power. Lastly, we evaluate the performance of the proposed MPC reformulation in simulation for a quadrupedal mobile manipulator, ALMA.

Keywords: Whole-Body Motion Planning and Control; Sensor-based Control; Legged Robots; Model Predictive Path Following Control, Mobile Manipulators.

Sommario

Questo lavoro formula un problema per la pianificazion di traiettorie con controllo a predizione del modello (MPC) di tipo sequenziale lineare quadratico (SLQ) per manipolatori a base fluttuante. Aggiungendo il parametro della curva agli stati del sistema, possiamo esercitare un controllo diretto sulla velocita' di progressione lungo la traiettoria. Sfruttiamo questo grado di libertà aggiuntivo per regolare la progressione lungo un percorso in modo da rispettare i vari costi e vincoli dinamici. Attraverso simulazioni su un manipolatore mobile Kinova a sette gradi di libertà, validiamo la nostra formulazione e dimostriamo la plasticità del risolutore in diversi scenari. Diverse situazioni sono studiate e presentate: una situazione nominale, una situazione per evitare collisioni e una situazione di recupero da un'inizializzazione errata. Viene inoltre fornito un breve studio qualitativo e quantitativo relativo ai risultati del caso nominale.

Questo lavoro introduce anche un nuovo approccio per la pianificazione del moto dell'intero corpo (whole-body motion) e la prevenzione di occlusioni dinamiche. L'approccio proposto riformula il vincolo di visibilità come una massimizzazione della probabilità di visibilità. In questa formulazione, aumentiamo la funzione di costo primaria di uno schema MPC per tutto il corpo attraverso una funzione barriera logaritmica rilassata (relaxed log-barrier) che produce una formulazione di massimizzazione di una probabilità logaritmica rilassata della probabilità di visibilità. La probabilità di visibilità è calcolata attraverso un campo d'ombra probabilistico che quantifica le occlusioni di sorgenti luminose puntuali. Forniamo inoltre gli algoritmi necessari per ottenere tale campo per entrambi i casi 2D e 3D. La tesi reporta implementazioni 2D di questo campo con simulazioni, e implementazioni 3D attraverso esperimenti hardware in tempo reale. Mostriamo che grazie al fatto che il nostro algoritmo di campo d'ombra ha una complessita' lineare rispetto alla dimensione della mappa, possiamo raggiungere una elevata frequenza di aggiornamento, il che facilita l'esecuzione a bordo di piattaforme mobili con potenza di calcolo limitata. Infine, valutiamo le prestazioni della riformulazione MPC proposta nella simulazione per un manipolatore mobile quadrupede, ALMA.

Parole chiave: Pianificazione e controllo del movimento dell'intero corpo (whole-body motion control); controllo basato sui sensori; robot su gambe; controllo per la pianificazione di traiettorie a predizione del modello, manipolatori mobili.

Notations

Nomenclature

ϵ	machine precision
η	central finite difference step size
n_l	number of links
n_j	number of joints
∇	gradient
n_q	number of actuated joints
n_b	number of base parameters
n_{b0}	minimum parameter number of n_b
$\boldsymbol{q}$	generalized coordinates vector
q_i	ith generalized coordinate
q_b	unactuated base coordinates
q_{n_j} n_j	actuated joint coordinates
q_{b_P}	translational base coordinates
q_{b_R}	rotational base coordinates
u	generalized coordinates speed
$\dot{u}$	generalized coordinates acceleration
T_{phi}	mapping similar to T_A
J_e	spatial Jacobian with respect to inertial frame
J	geometric Jacobian
J_A	analytical Jacobian
v_B	generalized base translational velocity
x_e	end-effector
r_e	end-effector position
ϕ_e	end-effector orientation
ϕ_{m_e}	minimal representation of end-effector orientation
$\dot{\phi}_i$ i_{th}	angular velocity
$\ddot{\phi}_i$ i_{th}	angular acceleration
$\dot{x}_e$	end-effector velocity in task space
$\dot{q}$	end-effector velocity in joint space
T_A	mapping used between minimal and non-minimal orientation representations of angular velocity
r, p, y	roll, pitch and yaw angle
$\dot{r}$, $\dot{p}$, $\dot{y}$	roll, pitch, yaw time-derivatives
T_e^I	transformation matrix between frames e, end-effector, and I, inertial frame

R_k^I	rotation matrix between frames k and inertial frame I
r_k^I	vector between frames k and I

Acronyms and Abbreviations

AD	Automatic Differentiation
ALMA	Articulated Locomotion and MAnipulation
BFGS	Broyden, Fletcher, Goldfarb, Shanno
CAD	Computer Aided Design
DOF	Degree of Freedom
DP	Dynamic Programming
FHOCP	Finite Horizon Optimal Control Problem
FOV	Field of View
GN	Gauss Newton
KKT	Karush-Kuhn Tucker triplet
LIDAR	Light Detection and Ranging
LP	Linear Program
LQ	Linear Quadratic
MPC	Model Predictive Control
MPFC	Model Predictive Path Following Control
NBV	Next-Best View
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
OCS2	Optimal Control for Switched Systems
PDQ	Probability-based Detection Quality
QP	Quadratic Program
ROS	Robot Operating System
SLQ	Sequential-Linear Quadratic
SQP	Sequential Quadratic Programming
UR	Universal Robot
URDF	Unified Robotic Description Format
WBC	Wholy-Body Controller

Operators

$$|x| = \sum_{i=1}^{n} |x_i| \qquad l_1 \text{ norm}$$

$$\|x\| = \sqrt{x^T x} = \sqrt{x_1^2 + \cdots + x_n^2} \qquad l_2 \text{ (Euclidean) norm}$$

$$a \times b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [a]_x b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_3 \\ -a_2 & -a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad \text{cross product/skew/unskew}$$

List of Figures

1.1 Mid-level Path Following MPC Architecture 4
1.2 Local Frenet-Serret Frame 4
1.3 MPFC based on Disturbance Observer for Wheeled Robots 5
1.4 Ballbot System Architecture 6
1.5 2D Visibility Constraints 6
1.6 Typical IBVS Control Architecture 7
1.7 Internal Model Control Structure for Predictive Visual Servoing . . . 7
1.8 A Fast Visibility Check based on Horizon Culling 8
1.9 Sampling Process in Next Best View Planning 8
1.10 Next Best View Planning 9
1.11 Probability-based Detection Quality Measure 10
1.12 Visibility and Obstruction in OpenAI's Multi-agent Hide and Seek . 10
1.13 Notions of Visibility and Obstruction: Visibility Polygons 10

2.1 ANYmal Floating Base Quadrupedal Robot 12
2.2 Kinova Gen3 7DOF with Vision Module 15
2.3 Kinova Gen3 7DOF Frame Definitions 16
2.4 Kinova Gen3 High-level and Low-level Servoing Control Architectures. 18
2.5 ALMA Composed of DynaArm Mounted on ANYmal C 19
2.6 Wireframe Illustration of ALMA Showing Main Frames and Forces . 20
2.7 Evolution of an Optimization Routine Solution 21
2.8 Sketch of Receding Horizon Strategy for Discrete-time MPC 24

3.1 Sketch of a 2D Geometric Path to be Followed 28
3.2 Obstacle Blocking a 2D Geometric Path to be Followed 31
3.3 Logarithmic Barrier Used in Interior-Point Methods 32
3.4 Relaxed Log Barrier 32
3.5 Wrapping the Shadow Field Cost with a Relaxed Log Barrier 34
3.6 Logit Scaling of the Orientation Penalty 35

4.1 Shadow Field Mapping Architecture 37
4.2 The Visibility Problem 38
4.3 2D Illustration Showing Pixels and Vectors of Interest 39
4.4 2D Illustration Showing Evolution of Pixels and Vectors of Interest . 40
4.5 Shadow Field 2D Weights Distributions 40
4.6 Hard Shadows Against Our Soft Shadows Approximation 41
4.7 2D Shadow Field Simulation 42
4.8 Black Voxel and its 3 Neighbours 43
4.9 XY, XZ, and YZ Views of Voxels 43
4.10 3D Cube and the Relevant Vectors and Planes 44
4.11 Local *Shadow Field* is a Subset of a Global Map 46
4.12 3D Shadow Field Simulation 47

5.1 Sample Reference Trajectory 50
5.2 Sample Reference Path to be Followed 51
5.3 Path Following Multiplicity Illustration in Nominal Case 52
5.4 Path Following Plots in Nominal Case 52
5.5 Path Following Multiplicity Illustration in Wrong Initialization Case 53
5.6 Path Following Plots in Wrong Initialization Case 53
5.7 Path Following Multiplicity Illustration in Collision Avoidance Case 54
5.8 Path Following Plots in Collision Avoidance Case 54
5.9 Path Following Quality in Nominal Case 55
5.10 Path Following Quality for X Component in Nominal Case 55
5.11 Path Following Quality for Y Component in Nominal Case 56
5.12 Path Following Quality for Z Component in Nominal Case 56
5.13 ALMA Control Architecture 57
5.14 Realtime *Shadow Field* Mapping Results 59
5.15 Sample Simulation Environments 60
5.16 Multiplicity Illustration Depicting Physics-based Simulation Results in Top View 61

A.1 ABB IRB 6400 71
A.2 Sample Filtering Scheme 77
A.3 A Fast Visibility Check for Handling Occlusions 79

B.1 Topview of a 2D Environment Containing Obstacles 81
B.2 Sideview of Fig. B.1 Showing Relevant Meshes 82
B.3 Illustration Showing Quiver Plot and Gradient-Based Solution . . . 82
B.4 Sideview of Evolution of our Mobile Kinova 7DOF in Simulation . . 83
B.5 Topview of Evolution of our Mobile Kinova 7DOF in Simulation . . 84
B.6 Frontview of Evolution of our Mobile Kinova 7DOF in Simulation . 85
B.7 Sideview of Evolution of our ALMA in Simulation 86
B.8 Topview of Evolution of our ALMA in Simulation 87
B.9 Frontview of Evolution of our ALMA in Simulation 88

List of Tables

2.1 Kinova Gen3 7DOF Denavit-Hartenberg Parameters 17
2.2 Kinova Gen3 7DOF Physical Quantities 17
2.3 Kinova Gen3 7DOF Inertial Parameters 17

List of Algorithms

1 *Shadow Field* Update Routine in 2D 41
2 *Shadow Field* Weights Routine in 3D 44
3 *Shadow Field* Update Routine in 3D 45

Chapter 1

Introduction

A mobile manipulator has a dual advantage of mobility offered by a mobile platform and agility provided by the manipulator. By extending the workspace of fixed-based robotic arms, mobile manipulation systems can access areas far from the reach of the ground-bolted counterparts. However, this mobility comes at the cost of more complex motion planning and navigation problems. Among these challenges, motion planning and visual tracking of a target point under inter-object occlusion remain partially unsolved.

Active vision methods have been a resort in the face of this challenge. Those methods often employ sampling in search of the next-best-view (NBV) or in order to perform view planning to maximize the quantity and quality of useful sensory data. However, active vision methods are still immature in terms of efficiency and efficacy and are still employed separately from motion planning itself, further increasing the complexity of the workflow.

On the other hand, implementing visual servoing in motion planning applications has limited success in preventing occlusions. In this setting, occlusions are avoided by constraining the camera pixel coordinates of the detected or tracked object of interest. This, in turn, restricts robot operation and control to the camera's Field of View (FOV).

At the core of occlusion avoidance lies the visibility problem. Geometrically, visibility is the existence of an unobstructed line of sight between the viewer and the object of interest, and it constitutes a necessary condition for any vision-based task such as detection, tracking, and manipulation. Currently, notions of visibility and obstruction are usually defined deterministically by utilizing ray-casting to simulate the line of sight. However, this approach only evaluates the visibility or lack of it in the scene, which is inadequate for gradient-based techniques and does not reflect reality with all of its uncertainties and sensory noise.

To this end, we propose a reformulation of the visibility constraint in an optimal control setting such that the likelihood of visibility is maximized. As such, we are solving the visibility problem within motion planning itself rather than doing it separately. To complement the reformulated visibility constraint, we also propose a novel method to evaluate notions of visibility and obstruction by building a probabilistic *shadow field* using a Dynamic Programming (DP) approach. This field is constructed using a probabilistic occupancy grid computed from RGB-D or

LIDAR data and centered around a point light source representing the target of interest that has to be tracked, detected, or manipulated. This field provides soft shadows, i.e., its values range from 0, representing cold regions, to 1, representing hot regions. Colder regions are regions lying in the shadow of the light source, i.e., it is less likely that a line of sight exists between those regions and the light source. On the contrary, hotter regions are regions where a line of sight to the light is more likely to exist. The light source itself can be either dynamic or static.

Even though constrained predictive path following has been investigated mostly for fixed base manipulators, it has also been studied for mobile robots ranging from applications to Unmanned Aerial Vehicles (UAV), to legged, bipedal, and humanoid robots and even autonomous underwater vehicles. The idea behind constrained path following in robotics is that we have a geometric path with no assigned speed, and we need to follow it optimally. Basically, since the timing of the path is unspecified, we have an additional degree of freedom in time, which we may exploit directly within motion planning like for example to halt progression along the path. This type of problems is most interesting for cases where a high precision in following the path is more important than the timing of the path, for example high precision welding or high precision machining. Constrained path following has been addressed in the literature using several solvers and formulations. We pose our own Sequential-Linear Quadratic (SLQ) Model Predictive Control (MPC) formulation in this work.

1.1 Outline and Contribution

The key contributions of this work are:

1. A path following SLQ-MPC problem formulation that exercises direct control over the rate of arclength progression along a path, therefore adding a degree of freedom in time.

2. Simulations on a Kinova 7 Degrees of Freedom (DOF) mobile manipulator validating our path following formulation in different scenarios.

3. A redefinition of the visibility constraint for optimal control that maximizes the likelihood of visibility in a manner suitable for whole-body motion planning methods.

4. A novel DP-based approach to describe the notions of visibility and obstruction, called *shadow field*, which is inherently probabilistic and efficient to calculate.

5. Hardware mapping tests validating our *shadow field*'s capability in capturing visibility and obstructions.

6. Physics-based simulations verifying our holistic occlusion avoidance implementation within motion planning of a mobile legged manipulator.

This thesis is structured as follows:

- **Chapter 1** introduces the major challenges tackled in this thesis and provides an overview of related work in the literature. It also provides a general description of the thesis structure and states explicitly the contributions of this work.

- **Chapter 2** provides a brief overview of the robot kinematics and dynamics for both *fixed base* and *floating base* systems, thus laying down the foundation for defining physically consistent kinematic and dynamic models for motion planning. In this chapter, the systems at hand are also introduced, the first of which is a mobile 7DOF Kinova manipulator that can be position, speed, or torque controlled, and the other is ALMA which is a DynaArm, a custom-made torque-controlled manipulator, mounted on a ANYmal, a quadrupedal legged robot. The latter platform will be addressed hereafter as ALMA. Lastly, this chapter briefly covers constrained numerical optimization for for optimal control problems (OCPs). The two main routines of interest are sequential quadratic programming (SQP) and sequential-linear quadratic (SLQ) control.

- **Chapter 3** poses the formulation of the OCPs that we are trying to solve. The problems are formulated as Finite Horizon Optimal Control Problems (FHOCPs) which are solved in an MPC fashion. Path following SLQ MPC is first proposed where we highlight the costs, equalities, and inequalities for the nominal case and for the case of collision avoidance. The formulation for dynamic occlusion avoidance within whole-body motion planning is then studied where we also pose our reformulation of the visibility constraint and discuss the additional system costs and constraints.

- In **Chapter 4**, *shadow field*, the field providing differentiable information about visibility and obstruction for the entire scene of interest, will be introduced. The visibility problem will be formally posed as well as our approach to solving it. 1D, 2D, and 3D simulations of *shadow field* will be presented and discussed. Moreover, the algorithms used to compute the *shadow field* be will be formally discussed, and a study on their computational and storage complexities will be presented.

- In **Chapter 5**, experimental and physics-based simulation results are presented. For the path following SLQ-MPC, we first provide a brief discussion on trajectory generation and obtaining a path as well as its arclength parametrization. We then put forward simulation results in three main cases; the nominal case, the case with wrong initialization, and the case with collision avoidance. For dynamic occlusion avoidance within whole-body motion planning, we first introduce the motion planning and control pipeline then we present the real-time hardware *shadow field* mapping results. Lastly, we conclude with physics-based simulations on ALMA.

- **Chapter 6** reiterates some weaknesses and missing links in both formulations and proposes potential future work directions as a continuation of the work in this thesis.

- In conclusion, **Chapter 7** summarizes the work presented in this thesis.

1.2 Literature Review

Path following NMPC, also known as Model Predictive Path Following Control, has been widely studied in the literature, most commonly applied to fixed base manipulators.

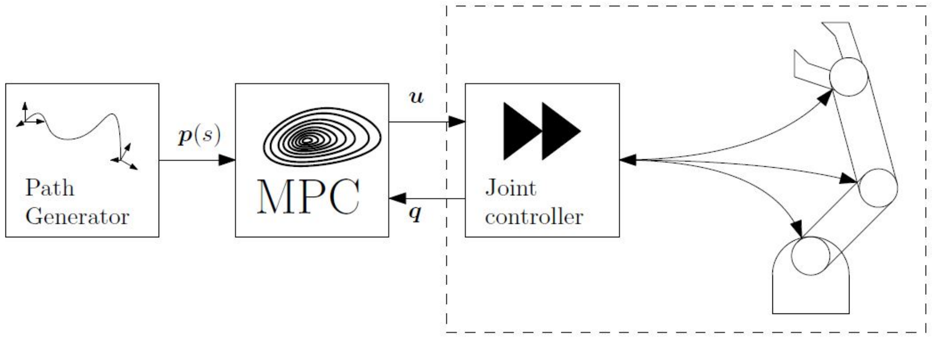


Figure 1.1: Architecture illustrated by [1] showing the mid-level MPC between the low-level joint controller and the higher level path generator. MPC plans a motion to follow the path provided by the generator and handles rudimentary runt-time constraints not considered by the generator, and passes commands to the low-level joint controller which provides feedback measurements to the MPC.

In [2], Arbo et al. show the MPFC's ability to stop at obstructions in a way that model predictive trajectory tracking cannot do. They adopt CasADi framework to define the symbolic expression of their MPC formulation and use a primal-dual interior point optimizer to solve their nonlinear program in a near real-time manner. In [1], Arbo et al. extend their previous work to a 6DOF manipulator where they implement their MPC as a mid-level controller between the low-level joint controller and the high-level path generator as depicted in Fig. 1.1. The mid-level controller handles rudimentary run-time constraints that are not considered by the path generator. They also compare their path following approach to a trajectory tracking one through experiments with UR5 and UR3 robots.

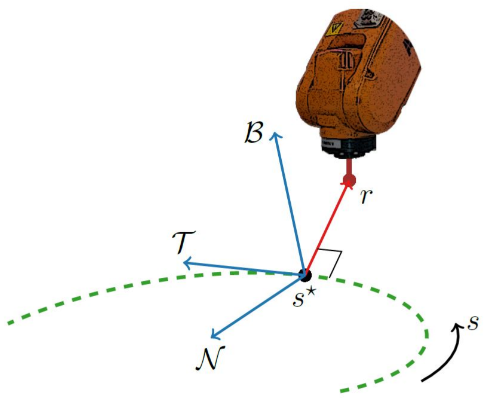


Figure 1.2: Orthonormal tangent, normal, and binormal unit vectors, $\mathcal{T}$, $\mathcal{N}$, and $\mathcal{B}$, respectively, describing the local Frenet-Serret frame as illustrated by [3]. s^* represents the optimal arclength value that minimizes the distance r between the end-effector and the path.

Duijkeren et al. [3] showcase their path following implementation in simulation for a 6DOF industrial robot that has to execute a writing task. They reformulate their problem from an unconstrained optimization routine that returns the optimal arclength along the path, which minimizes the distance between current end-effector position in inertial world frame and the path, to a highly nonlinear spatial adaptation. On the other hand, in this reformulation, knowledge about the temporal evolution of vectors describing the local Frenet-Serret frame, Fig. 1.2, at time t is explicitly available in the integration scheme for derivative computation, and geometric constraints transform into simple linear or norm bound constraints. Their approach is more suitable for static constraints than time-varying ones.

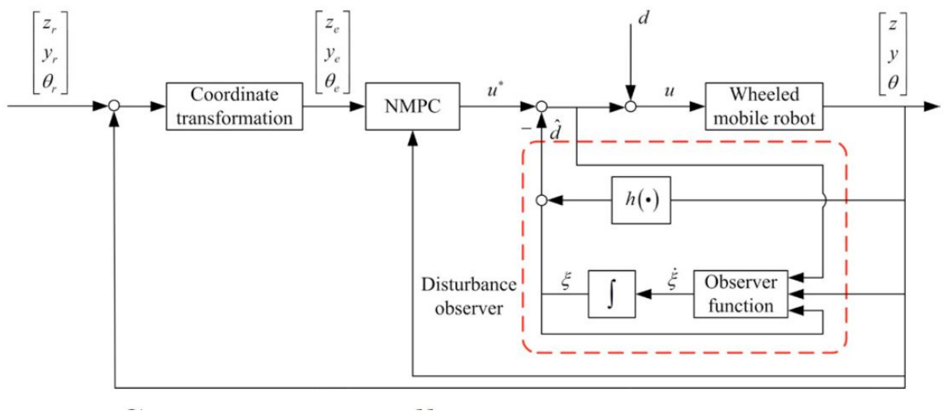


Figure 1.3: Path following MPC architecture based on a side slip disturbance observer for wheeled robots as illustrated by [4]. Error in coordinates is obtained from the feedback loop and fed to the NMPC together with the observed disturbance. Optimal control input is then applied with disturbance to the wheeled robot.

Path following has been also investigated for wheeled mobile robots. In [5], it is shown that path following problems of wheeled mobile robots to a given path may be reduced to an error regulation problem. A robust model predictive control based on disturbance observer is proposed as a solution against the the potential significant degradation of path following performance arising due to side slip, which may even cause instability, and is depicted in Fig. 1.3.

Faulwasser et al. [6] perform a path following task, depicted on the left in Fig. 1.4, for both problems with no speed assignment and problems with speed assignment. Trajectory tracking and speed-assigned path following are differentiated where in trajectory tracking there is only speed path to be followed but in the speed-assigned path following the problem admits several reference trajectories differing with respect to the path parameter. In the latter, the approach is similar to scaling the timing law on the go within motion planning, whereas it is more challenging to do in standard trajectory-tracking formulations.

Even though path following is most commonly applied to mobile and fixed-base manipulators, there exist recent investigations for underactuated ballbots, one of which is depicted on the right in Fig. 1.4. Jespersen et al. [4] prove that highly nonlinear underactuated ballbot system can be controlled as a linear system which does not even include the underactuated properties. Deviations in time from the desired trajectories are allowed, hence the underactuated deviations are allowed without penalties.

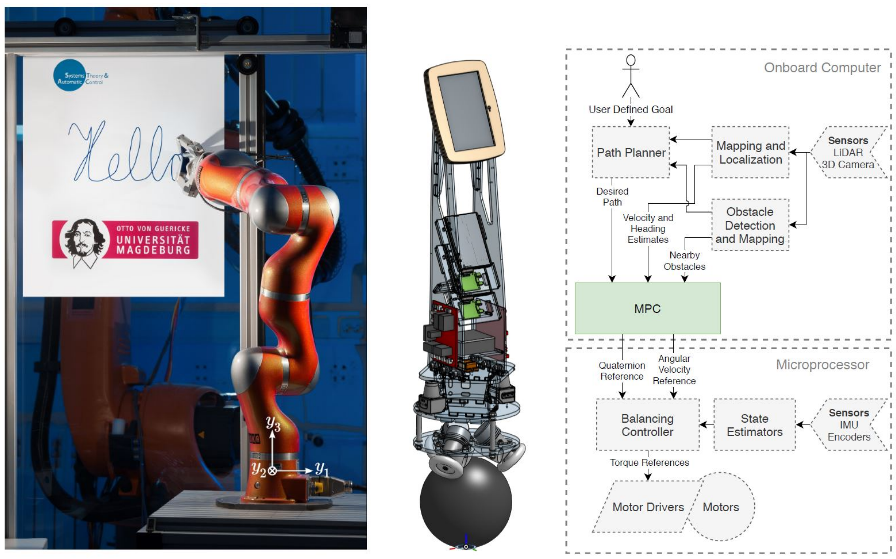


Figure 1.4: Generic writing task performed by a fixed-base manipulator as illustrated by [6] on the left and the ballbot path following MPC system architecture as illustrated by [4] on the right. For the latter, perceptive and navigation data are fed to the MPC which commands the balancing controller. The balancing controller itself is in a high-bandwidth loop with the sensors, and commands the motor drives.

Visibility constraints, also known as two-dimensional constraints, are often used in MPC schemes to keep the image-plane coordinates of interest within the camera's FOV [7, 8, 9] as can be seen in Fig. 1.5. Such constraints can also represent forbidden regions in the image.

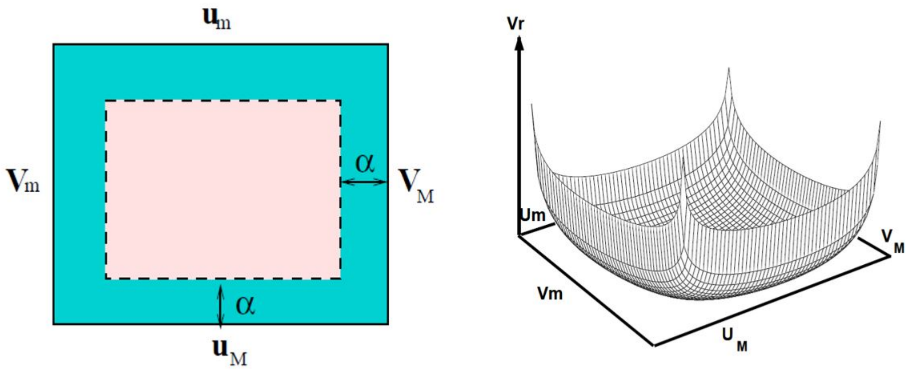


Figure 1.5: 2D Visibility constraints in optimal control methods work on limiting the camera pixel coordinates u and v of the target of interest within bounds as can be seen on the left. On the right, a relaxation of this hard constraint is visualized where pixel coordinates are repelled from extremities of the camera field of view and attracted towards its center. Illustrated by [10].

Using this type of constraint has some limits. For example, control, in this case, is typically interrupted if the target goes out of FOV [9]. Mezouar and Chaumette [10] use a softened visibility constraint which is reformulated as a repulsive potential field, but such a constraint highly increases the chance of having local minima in the overall potential field [11]. One further limitation of adopting visibility constraints, asides from the fact that control needs to be halted once vision of the target is lost, is the need to work within a visual-servoing framework as the ones illustrated in Fig. 1.6 and Fig. 1.7, and the need for a camera sensor.

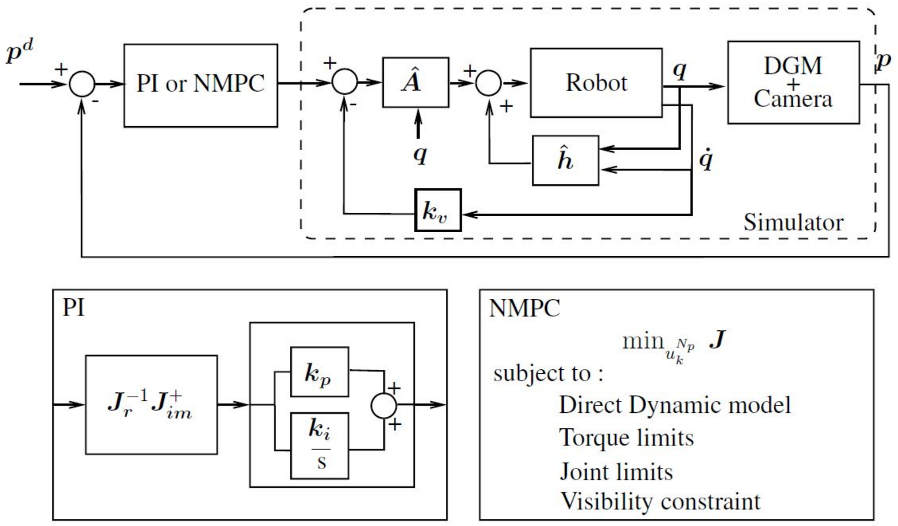


Figure 1.6: Typical control scheme for robot manipulator image-based visual servoing. Camera model is linearized and either an NMPC or a classical PI controller may be used. Illustrated by [8].

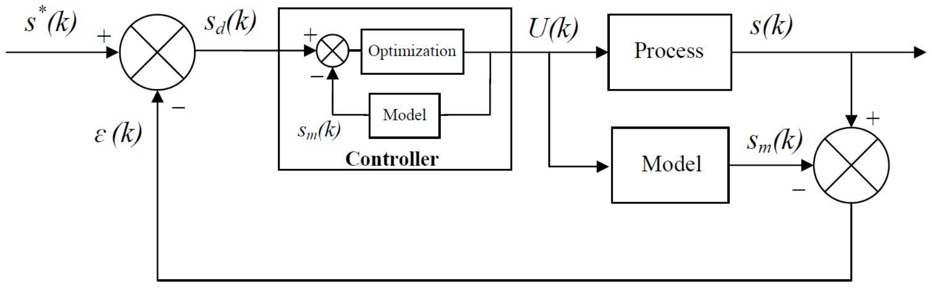


Figure 1.7: Well-known internal model control structure adopted and illustrated by [9]. $\mathcal{E}$ represents all modelling errors and disturbances between current image features and predicted ones from the model. The classical controller is replaced by a predictive optimization algorithm minimizing the image feature errors.

In [12], Zhang et al. perform dynamic self-occlusion avoidance optimally, but indirectly, based on the depth image sequence of moving objects. Objects are first reconstructed then their motion is estimated by matching two Gaussian curvature feature matrices. An optimal planner then decides the camera motion that will lead to avoiding self-occlusion. Self-occlusions, as well as mutual occlusions for the case of a multi-arm robotic cell equipped with several cameras, are predicted in [13] where the geometric model of the objects is assumed to be known. Recovery of a UAV target tracker from detection discontinuities caused by occlusion in a dynamic environment is addressed in [14] without tackling occlusion in itself.

Nageli et al. [15] implement occlusion minimization within an MPC scheme.

This method, however, is limited to evaluating a single point against an ellipsoidal approximation of obstacles using a fast visibility check[1].

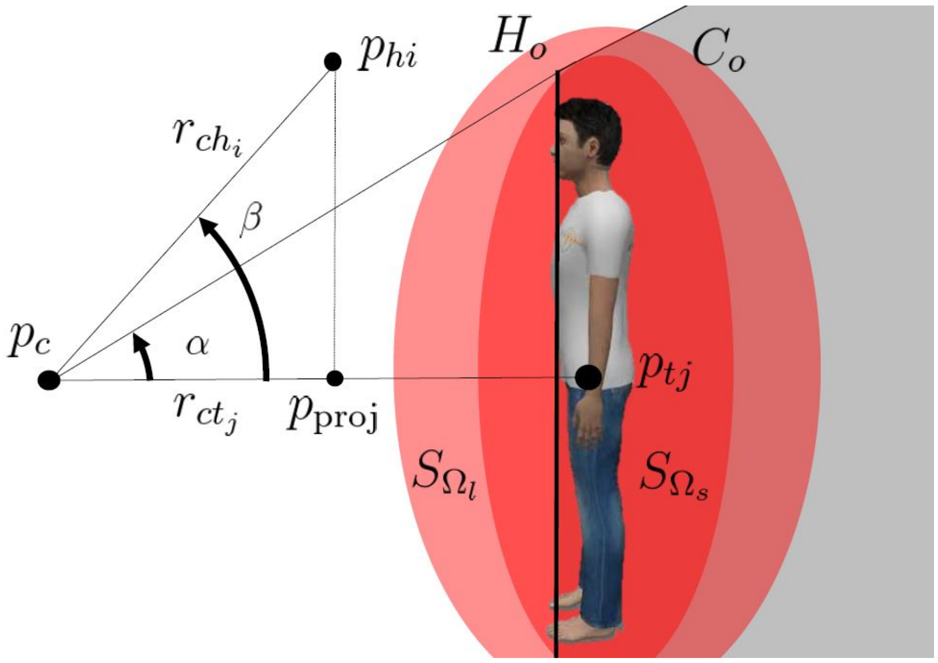


Figure 1.8: Fast and simple geometric visibility check based on ellipsoidal approximation of occluding. Illustrated by [16]. Based on horizon culling, this method tests if the projection of point of interest lies beyond the horizon H_o or not.

Most NBV planning methods are still immature in terms of efficiency and are nowhere near real-time. Wu et al. [17] perform NBV planning by evaluating the visibility as well as the likelihood of feature matching, achieving around 1.5s per step with the framework depicted in Fig. 1.9.

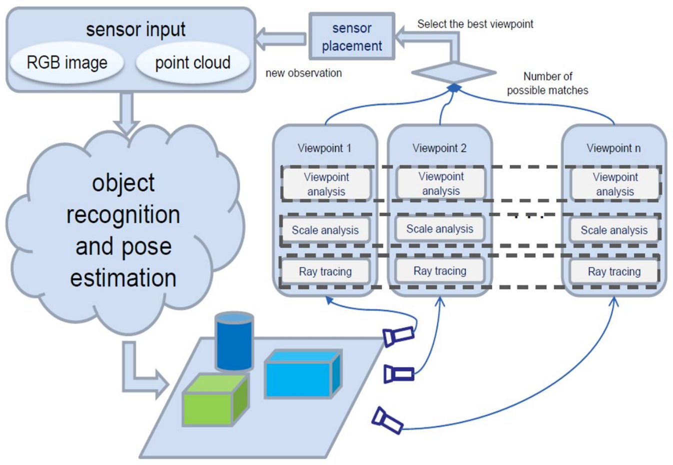


Figure 1.9: The sampling process in search of the view that maximizes object recognition [17]. The optimal position is fed to the planner in a separate step.

[1]Check Appendix A for details

On the other hand, [16] plans the NBV by utilizing online aspect graphs to account for occlusions and feature visibility, requiring more than a minute of planning per movement. This framework can be seen in Fig. 1.10.

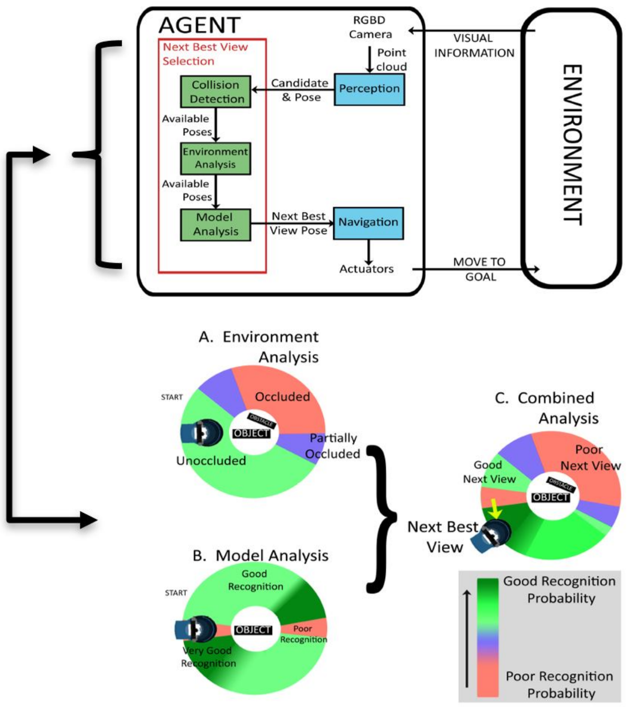


Figure 1.10: NBV Planning aims to optimize the positioning of the camera sensor so as to maximize some criteria, including visibility (or occlusion avoidance). The resulting position is then fed to the motion planner in a separate step. Illustrated by [16].

NBV methods often utilize ray-casting [17, 16, 18]. Ray-casting is widely used in robotic planning applications, for example, in autonomous scene exploration [19] where a dual OcTree structure is used to encode regions which are occupied, free, and unknown, that are then explored via an NBV planning approach. Typically used methods to simulate ray-casting are voxel traversal algorithms such as [20]. There exist learning-based methods to evaluate uncertainties arising from image-based detection [21, 22]. Such methods, like the one represented Fig. 1.11, are a-posteriori and are incapable of providing meaningful predictive power for optimal control methods.

The notions of visibility and obstruction are of great interest in multi-agent hide-and-seek or pursuit-evasion applications. In [23], a vision cone is constructed for each agent using a LIDAR-like array of 30 rays, as seen in Fig. 1.12, and any agent not within the line of sight is masked. Tandon and Karlapalem [24] represent the notion of visibility in simulation as an associated visibility region for each agent, which itself is also constructed by tracing uniformly spaced rays emitted from the agent. Moreover, the visibility problem is a basic problem in computational geometry [25, 26, 27] and has applications in computer graphics [28].

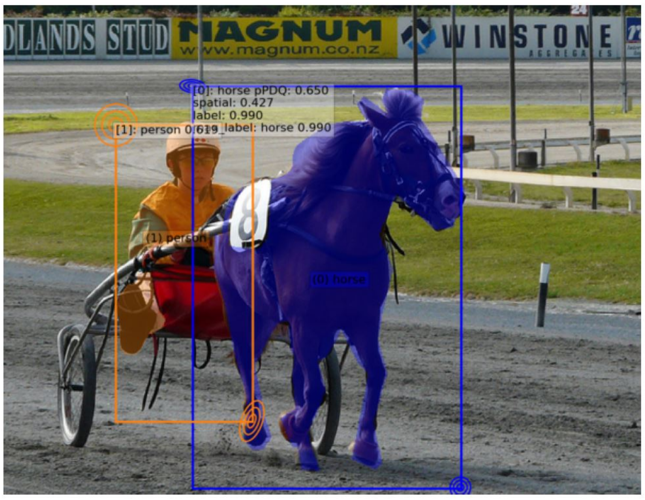


Figure 1.11: A-posteriori qualification measure of detection, pPDQ, in [22], of a person and a horse based on criteria including visibility and occlusion.

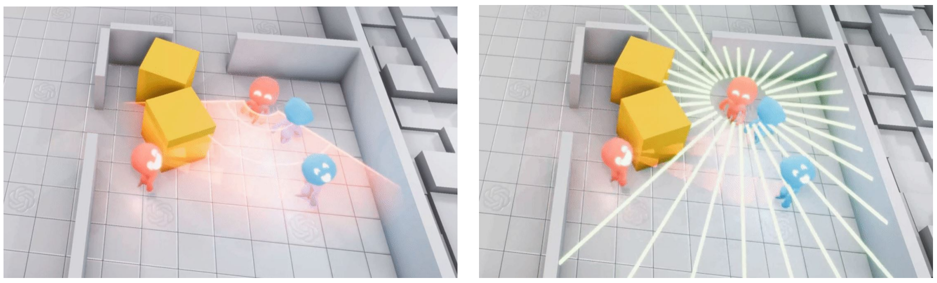


Figure 1.12: OpenAI's multi-agent hide and seek. Each seeking agent has a simulated LIDAR which casts 30 rays in all directions. Those rays are used to define visibility field of view and whether hiding agents are within that field of view or not.

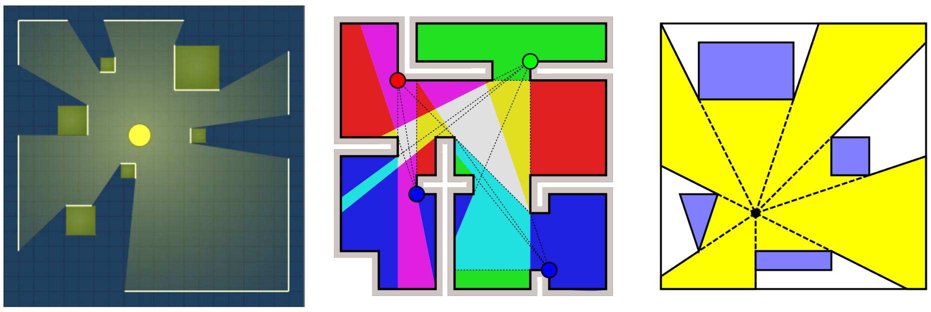


Figure 1.13: Notions of visibility and obstruction are most commonly defined using visibility polygons. Rays are cast in all directions, defining regions of visibility based on collisions of rays with objects. From left to right: light source and resulting shadows cast by the obstacles, art gallery (place guards to achieve full visibility of the art gallery), visibility polygons in a 2D environment[2].

[2]The first illustration in Fig. 1.13 is provided by https://www.redblobgames.com/articles/visibility/, the other two illustrations are provided by Wikipedia user Claudio Rocchini.

Chapter 2

Preliminaries and System Overview

In this chapter, kinematic and dynamic models are briefly introduced, based on [29], [30], [31], and [32], for two different types of robotic systems, namely *fixed base* and *floating base* systems, so as to setup the context of the formulations to be put forward in **Chapter 3**. Fixed base systems are systems where the root link frame is static, often fixed to the world inertial frame. This is the case with most industrial manipulators for example. On the other hand, floating base robots like mobile manipulators, quadrupeds, or humanoids, can move freely, unconstrained to the world frame. As such, we will be laying down the foundations for defining physically consistent kinematic and dynamic models in motion planning. More specifically, we briefly cover the necessary background on robot forward kinematics and the Jacobian as we will be performing kinematic MPC for path following. We also briefly derive the well-known robot dynamic model which will be necessary for performing whole-body motion planning.

Moreover, we introduce our platforms, the mobile Kinova 7DOF and ALMA, both of which being floating base systems, and present the centroidal dynamics formulation for ALMA. At the end of each robot's section, the respective state and input vectors are presented.

Lastly, we provide an overview about constrained numerical optimization for continuous-time OCPs based on [33]. Even though we do not use SQP to solve our formulations, we provide SQP's main concepts for the reader to get the gist of using numerical optimization in optimal control methods. This paves the way to put forward SLQ control and consequently SLQ MPC, which is the main routine of interest in this work.

2.1 Kinematics

Kinematics is the study of motion, i.e., positions, speeds, and accelerations, of a point or an object, or a system of points of objects, irrespective of forces causing of the motion. For points in 3D, a description in terms of position $\in \mathbb{R}^3$ is sufficient. On the other hand, 3D bodies require additional description of their orientations $\phi \in \mathrm{SO}(3)$ with respect to a reference frame.

Typically, robotic systems are modeled as an open chain of bodies composed of $n_l = n_j + 1$ links where n_j is the number of joints connecting them, which may be either prismatic or revolute. Revolute joints provide a rotational degree of freedom q_i, whereas for a prismatic joint q_i represents a linear displacement.

Fixed Base Kinematics

Classical fixed based kinematics are reported in Appendix A based on the sources cited in this chapter's introduction.

Floating Base Kinematics

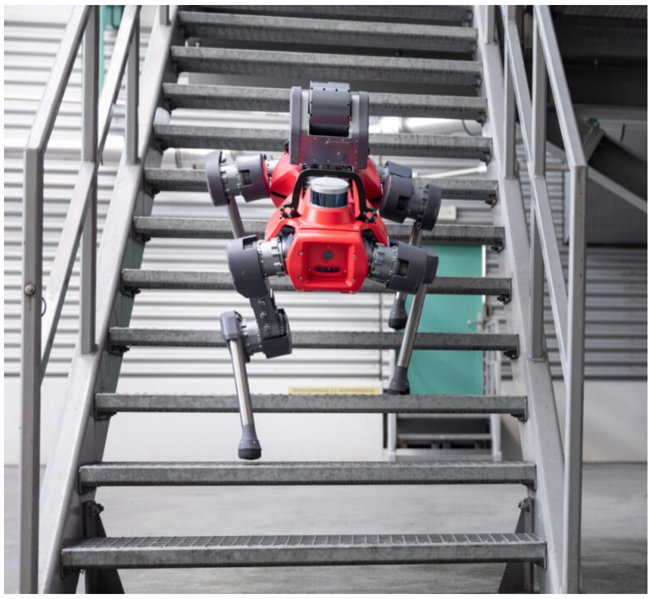


Figure 2.1: Floating base quadrupedal robot, ANYmal, climbing stairs. Source: ANYbotics.

Now for a floating base robot, like the quadruped ANYmal robot depicted in Fig. 2.1 or humanoid robots, generalized coordinates are composed of n_b un-actuated base coordinates $\mathbf{q}_b$ and n_j actuated joint coordinates $\mathbf{q}_j$

$$\mathbf{q} = \begin{pmatrix} \mathbf{q}_b \\ \mathbf{q}_{n_j} \end{pmatrix}, \tag{2.1}$$

where the base itself is in free translation and rotation,

$$\mathbf{q}_b = \begin{pmatrix} \mathbf{q}_{b_P} \\ \mathbf{q}_{b_R} \end{pmatrix} \in \mathbb{R}^3 \times SO(3), \tag{2.2}$$

noting that $\mathbf{q}_{b_P}$ and $\mathbf{q}_{b_R}$ may not necessarily be parametrized as in (2.2), whereby the minimum parameter number n_{b0} is 6.

Furthermore, the generalized velocity and acceleration vectors are introduced since differentiation in SO(3) is different from $\mathbb{R}^3$,

$$\mathbf{u} = \begin{pmatrix} \mathbf{v}_B \\ \omega_B \\ \dot{\phi}_1 \\ \vdots \\ \dot{\phi}_{n_j} \end{pmatrix} \in \mathbb{R}^{6+n_j} = \mathbb{R}^{n_u}, \qquad \dot{\mathbf{u}} = \begin{pmatrix} \mathbf{v}_B \\ \psi_B \\ \ddot{\phi}_1 \\ \vdots \\ \ddot{\phi}_{n_j} \end{pmatrix}, \tag{2.3}$$

where $\mathbf{u}$ may be obtained as,

$$\mathbf{u} = T_\phi \cdot \dot{\mathbf{q}}, \text{ with } T_\phi = \begin{bmatrix} \mathbb{I}_{3\times3} & 0 & 0 \\ 0 & T_{\mathbf{x}_R} & 0 \\ 0 & 0 & \mathbb{I}_{n_j\times n_j} \end{bmatrix}, \tag{2.4}$$

whereby $T_{\mathbf{x}_R}$ depends on the chosen representation/parametrization of rotation.

The end-effector $\mathbf{x}_e$ of a floating base robot has the position vector

$$r_e^I(\mathbf{q}) = r_B^I(\mathbf{q}) + R_B^I(\mathbf{q}) \cdot r_e^B(\mathbf{q}), \tag{2.5}$$

with respect to the inertial frame where r_B^I is the position of the base frame with respect to the inertial frame and, $R_B^I(\mathbf{q})$ is the rotation of the base frame in the inertial frame, and $r_e^B(\mathbf{q})$ is the position of the end-effector with respect to the base frame.

In order to obtain the differential forward kinematics, we differentiate (2.5) with respect to time, allowing us to obtain the spatial Jacobian with respect to the inertial frame

$$J_e(\mathbf{q}) = \begin{bmatrix} J_P \\ J_R \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{3\times3} & -R_B^I \cdot [r_e^B]_\times & R_B^I \cdot J_{P_{q_j}}(\mathbf{q}_j) \\ 0_{3\times3} & R_B^I & R_B^I \cdot J_{R_{q_j}}(\mathbf{q}_j) \end{bmatrix}, \tag{2.6}$$

which realizes the mapping from generalized velocities $\mathbf{u}$ to operational space twist of the end-effector frame with respect to inertial frame

$$\begin{bmatrix} v_e^T & \omega_e^T \end{bmatrix}^T. \tag{2.7}$$

Kinematic Redundancy

A robot is said to be kinematically redundant if it has more degrees of freedom than those strictly necessary to perform a task, i.e., if $n_j > m$ where m is the dimension of our task space. As such, redundancy is task-dependent where m may actually be less than the dimension of the operational space. For example, controlling the position and orientation of a manipulator end-effector $\mathbf{x}_e \in \mathbb{R}^3 \times SO(3)$ with a 7DOF manipulator is said to be redundant task, as $n_j = 7 > m = 6$.

Kinematic redundancy is typically exploited to:

- increase dexterity and manipulability

- avoid obstacles
- avoid kinematic singularities
- minimize energy consumption
- increase safety

2.2 Dynamics

Dynamic robot models allow us to describe how and why will forces alter a system and its motion, providing us with a powerful tool to predict, simulate, and control the behaviour of the robot. There are several approaches to obtain the equations of motion governing the dynamics of a robot, but they are all mainly based on classical Newtonian and Lagrangian mechanics and they all result in equivalent models. The first of those methods is the recursive *Newton-Euler* method. It performs two sweeps of computations, one forward sweep propagating the velocities and accelerations starting from the base frame to the end effector, and one backward sweep propagating forces and moments from end-effector to base frame. Its iterative nature makes it computationally efficient and suitable for online applications. The second technique is *Euler-Lagrange*. Even though it is analytical and requires more computations, it is possible to obtain explicit expressions of the matrices of the equations of motion as a function of the generalized coordinates, generalized velocities, and generalized accelerations, allowing very efficient evaluations of the equations of motion in real-time. In this thesis, *Euler-Lagrange* will be first briefly described assuming a fixed base system with n_q joints and dynamic bodies. Floating base dynamics will then be presented in a separate section.

Euler-Lagrange Dynamics

Derivation of Euler-Lagrange dynamics for a fixed base system is reported in Appendix A.

Dynamics of Floating Base Systems

Deriving the dynamics for a floating base system is fairly similar to fixed base systems, wit the exception of using the generalized coordinates in (2.1) and respectively the generalized velocities and accelerations in (2.3). Moreover, we add the selection matrix S which specifies which of our joints are actuated according to

$$\mathbf{u}_j = \mathbf{S}\mathbf{u} = \mathbf{S}\begin{pmatrix}\mathbf{u}_b \\ \mathbf{u}_j\end{pmatrix} = \begin{bmatrix}0_{6\times 6} & \mathbb{I}_{6\times n_j}\end{bmatrix}\begin{pmatrix}\mathbf{u}_b \\ \mathbf{u}_j\end{pmatrix} \tag{2.8}$$

where $u_j = \dot{q}_j \in \mathbb{R}^{n_j}$ and $u_b \in \mathbb{R}^6$.

Unacuated base coordinates $\mathbf{q}_b$, however, are controlled using *external forces* F_{ext} arising for example from contacts for land robots or aerodynamics for flying robots.

Omitting the explicit friction terms from (A.28), and taking into consideration the aforementioned comments, we may write the equation of motion for our floating base system as

$$\boldsymbol{B}(\mathbf{q})\dot{\mathbf{u}} + \boldsymbol{C}(\mathbf{q},\mathbf{u}) + \boldsymbol{g}(\mathbf{q}) = \mathbf{S}^T\boldsymbol{\tau} + \mathbf{J}_{ext}^T\mathbf{F}_{ext} \tag{2.9}$$

with its components being:

$\mathbf{B}(\mathbf{q})$	$\in \mathbb{R}^{n_q \times n_q}$	Generalized mass matrix (orthogonal).
$\mathbf{q}, \mathbf{u}, \dot{\mathbf{u}}$	$\in \mathbb{R}^{n_q}$	Generalized coordinates, velocity, and acceleration.
$\mathbf{C}(\mathbf{q}, \mathbf{u})$	$\in \mathbb{R}^{n_q}$	Coriolis and centrifugal terms.
$\mathbf{g}(\mathbf{q})$	$\in \mathbb{R}^{n_q}$	Gravitational terms.
$\mathbf{S}$	$\in \mathbb{R}^{n_\tau \times n_q}$	Selection matrix of actuated joints.
$\boldsymbol{\tau}$	$\in \mathbb{R}^{n_q}$	Generalized torques acting in direction of generalized coordinates.
$\mathbf{F}_{ext}$	$\in \mathbb{R}^{n_c}$	External acting forces (e.g. from contacts).
$\mathbf{J}_{ext}$	$\in \mathbb{R}^{n_c \times n_q}$	Geometric Jacobian of location where external forces apply.

2.3 Kinova 7DOF Mobile Manipulator

Figure 2.2: Kinova Gen3 7DOF with the vision module attached to the end-effector as provided by Kinova Robotics.

Kinova Gen3 7DOF is the system with which we carried out simulations to validate our path following formulation. What makes this manipulator special is the open-source nature of its documentation. Unlike traditional practices by companies like ABB, Kinova provides the complete robot specifications including all the necessary kinematic and dynamic parameters to build precise models of the arm. This makes this robot particularly more oriented for research and educational purposes. As a matter of fact, the Kinova Gen3 is a lightweight, adaptable, and modular arm that allows easy integration within ROS and MATLAB and Simulink thanks to the Kortex API software provided with it. Moreover, Kinova offers a few end-effector interface modules like a vision module and a gripper module. More details about this manipulator are present in the technical sheet in Appendix C.

All frame and joint placements, Denavit-Hartenberg parameters, physical quan-

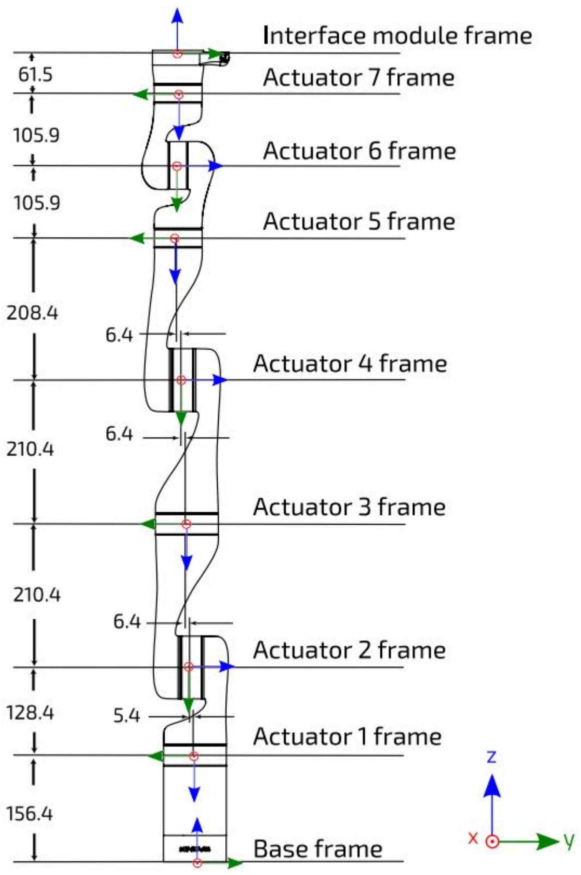


Figure 2.3: Kinova Gen3 7DOF frame definitions.

tities (mass and center of mass (CoM)), and inertia tensors are displayed in Fig. 2.3 and Tables 2.1, 2.2, and 2.3 respectively.

Moreover, Kinova also provides Computer Aided Design (CAD) models of each body in the manipulator. Using the provided kinematic and dynamic parameters and the CAD models, we are able to construct a detailed description of the robot in form of a Unified Robotic Description Format (URDF), allowing us to use the robot in Robot Operating System (ROS) and in physics-based simulation environments such as Gazebo. This format also allows us to use Pinocchio, a state-of-the-art rigid body dynamics solver for poly-articulated systems [34]. Pinocchio is an open-source project that is tailored for robotic applications and is built using Eigen, an efficient linear algebra library. Pinocchio performs forward kinematics, forward and inverse dynamics, centroidal dynamics, as well as all of their analytical derivatives in a multi-thread friendly way. Pinocchio also supports automatic differentiation frameworks, which is important for MPC implementations.

Kinova Gen3 robot control can be easily integrated with ROS and MATLAB and Simulink due to the Kortex API software that Kinova provides. The arm may be servo-controlled in position, speed, acceleration, or torque modes at either the high level or the low level, as illustrated in Fig. 2.4.

In simulation, we mount this manipulator on a mobile base that has 3 degrees of freedom in xy plane, being movement along x direction, movement along y direction,

Table 2.1: Kinova Gen3 7DOF Denavit-Hartenberg parameters.

i	$\boldsymbol{\alpha_i}$ **(radians)**	$\boldsymbol{a_i(m)}$	$\boldsymbol{d_i(m)}$	$\boldsymbol{\theta_i}$ **(radians)**
0 (from base)	π	0.0	0.0	0
1	$\frac{\pi}{2}$	0.0	$-(0.1564 + 0.1284)$	q_1
2	$\frac{\pi}{2}$	0.0	$-(0.0054 + 0.0064)$	$q_2 + \pi$
3	$\frac{\pi}{2}$	0.0	$-(0.2104 + 0.2104)$	$q_3 + \pi$
4	$\frac{\pi}{2}$	0.0	$-(0.0064 + 0.0064)$	$q_4 + \pi$
5	$\frac{\pi}{2}$	0.0	$-(0.2084 + 0.1059)$	$q_5 + \pi$
6	$\frac{\pi}{2}$	0.0	0.0	$q_6 + \pi$
7 (to interface)	π	0.0	$-(0.1059 + 0.0615)$	$q_7 + \pi$

Table 2.2: Kinova Gen3 7DOF physical quantities.

Link Segment	**mass (kg)**	**CoM coordinates (m)**
Base	1.697	[-0.000648, -0.000166, 0.084487]
1	1.377	[-0.000023, -0.010364, -0.073360]
2	1.1636	[-0.000044, -0.099580, -0.013278]
3	1.1636	[-0.000044, -0.006641, -0.117892]
4	0.930	[-0.000018, -0.075478, -0.015006]
5	0.678	[0.000001, -0.009432, -0.063883]
6	0.678	[0.000001, -0.045483, -0.009650]
7 (without vision module)	0.364	[-0.000093, 0.000132, -0.022905]
7 (with vision module)	0.500	[-0.000281, -0.011402, -0.029798]

Table 2.3: Kinova Gen3 7DOF inertial parameters ($kg \cdot mm^2$).

Link Segment	I_{xx}	I_{xy}	I_{xz}	I_{yy}	I_{yz}	I_{zz}
Base	4622	9	60	4495	9	2079
1	4570	1	2	4831	448	1409
2	11088	5	0	1072	-691	11255
3	10932	0	-7	11127	606	1043
4	8147	-1	0	631	-500	8316
5	1596	0	0	1607	256	399
6	1641	0	0	410	-278	1641
7 (without vision module)	214	0	1	223	02	240
7 (with vision module)	587	587	3	369	118	609

and yaw orientation of the base about the z-axis representing the robot base heading. This results in a state space dimension of $n_x = 3 + n_j = 3 + 7 = 10$, where n_j is the number of degrees of freedom enabled by the manipulator joints. Setting the derivatives of those degrees of freedom as our control input means an input of dimension $n_u = 10$, but we may exercise control over the norm of the velocity of the robot base directly, allowing us to reduce that number from 10 to 9. As such, our state and input vectors can be respectively written as

$$\boldsymbol{x} = \begin{bmatrix} b_x & b_y & b_\psi & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 \end{bmatrix}, \tag{2.10}$$

$$\boldsymbol{u} = \begin{bmatrix} v_b & \dot{\psi} & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 & \dot{q}_4 & \dot{q}_5 & \dot{q}_6 & \dot{q}_7 \end{bmatrix}. \tag{2.11}$$

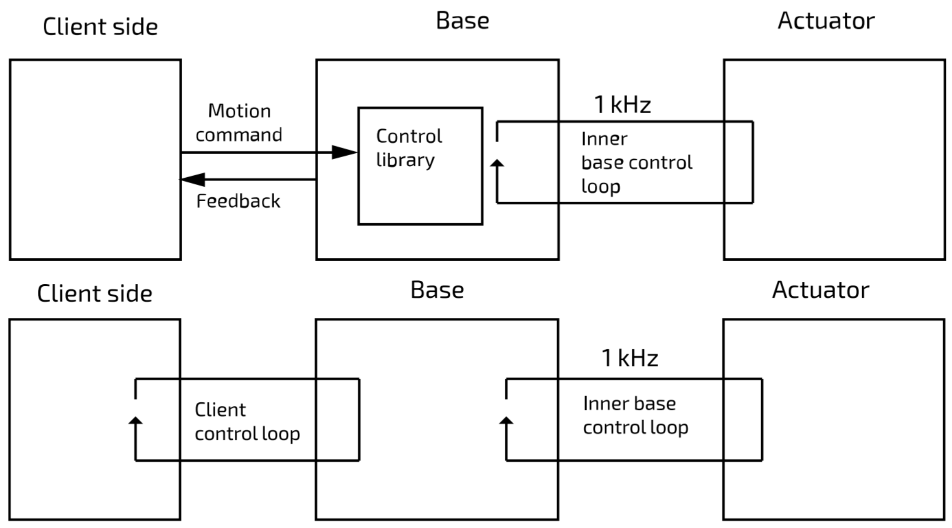


Figure 2.4: Kinova-provided high-level and low-level servoing control architecture schemes. In the high-level servoing mode allows the user to send position or velocity commands to the robot via an API at low frequency. The control library handles direct and inverse kinematics as well as applies safety limits. At the low level, the user has to handle those manually, then pass the command directly to the actuators. The base control loop runs at 1kHz communication.

Since we do not perform torque control in path following, the dynamic model is not of relevance for Kinova, but it can nonetheless be adopted directly using (A.28).

2.4 Articulated Locomotion and MAnipulation - ALMA

ALMA is a robot composed of a custom-made, torque-controllable 4 degrees of freedom robotic arm, DynaArm, mounted on a quadrupedal platform, ANYmal C. The version of ALMA we study is the one first introduced in [35]. We highlight the position of the Velodyne sensor and the robot end-effector in ALMA's standing position in Fig. 2.5. ALMA is equipped with four Realsense D435 cameras that provide 360° vision as well as a VLP16 Puck LITE Velodyne sensor mounted on the robot's rear. It has two on-board computers, one running locomotion and planning modules, and the other running mapping modules.

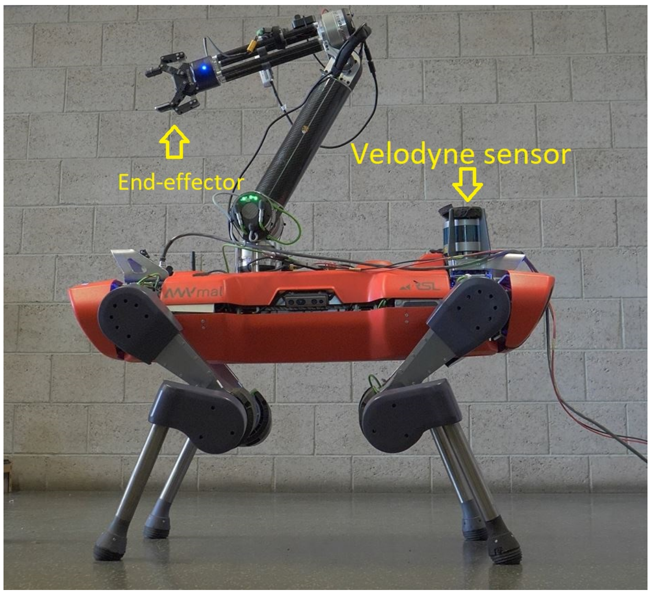


Figure 2.5: Photo showing ALMA, a robot composed of a custom-made, torque-controllable 4 degrees of freedom robotic arm, DynaArm, mounted on a quadrupedal platform, ANYmal C. We highlight the position of the Velodyne sensor and the robot end-effector in ALMA's standing position.

We adopt the system dynamic modelling provided in [35]. Basically, in reference to the dynamic model for floating base systems (2.9), we may describe ALMA with the system of equations

$$\boldsymbol{B}_u(\mathbf{q})\dot{\mathbf{u}} + \boldsymbol{C}_u(\mathbf{q}, \mathbf{u}) = \mathbf{J}^T_{ext_u}\mathbf{F}_{ext}, \tag{2.12a}$$

$$\boldsymbol{B}_a(\mathbf{q})\dot{\mathbf{u}} + \boldsymbol{C}_a(\mathbf{q}, \mathbf{u}) = \boldsymbol{\tau}_a + \mathbf{J}^T_{ext_a}\mathbf{F}_{ext}, \tag{2.12b}$$

where we split the dynamic model to unactuated and actuated dynamics respectively corresponded to by the subscripts u and a (previously b and n_j respectively in Section 2.1). We also collapsed the gravitational terms together with the Coriolis and centrifugal terms. Moreover, we are adopting a ZYX-Euler angle parametrization to represent the base's orientation. The external forces acting on the robot's limbs are are depicted in Fig. 2.6. Sleiman et al. [35] assume that the subsystem (2.12a) may be independently studied, from which one could equivalently retrieve the Newton-Euler equations applied to the robot's CoM, or the centroidal dynamics, by a proper transformation, yielding

$$\dot{h}_{com} = \begin{bmatrix} \sum_{i=1}^{n_c} \mathbf{f}_{c_i} + mg \\ \sum_{i=1}^{n_c} \mathbf{r}_{com,c_i} \times \mathbf{f}_{c_i} + \boldsymbol{\tau}_{c_i} \end{bmatrix}, \tag{2.13}$$

where $\dot{h}_{com} = (p_{com}, l_{com}) \in \mathbb{R}^6$ represents the centroidal momentum composed of linear and angular momentum about the centroidal frame $\mathcal{G}$ which is aligned with the world frame but attached to the robot's CoM. n_c is the number of contact forces and torques applied by the environment on the robot at contact points c_i with positions r_{com,c_i} with respect to the CoM.

The generalized coordinates' rate of change affects the centroidal momentum based on the centroidal momentum matrix mapping $\mathbf{A}(\mathbf{q}) \in \mathbb{R}^{6\times(6+n_a)}$. This mapping may be obtained as a function of the full kinematic configuration and the

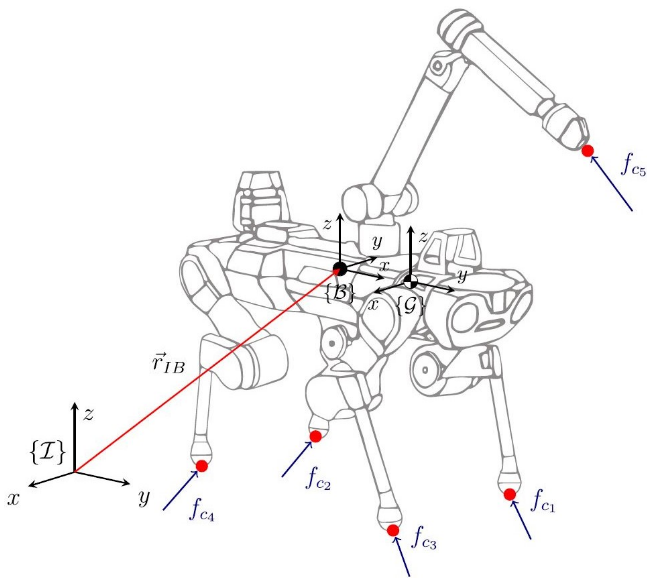


Figure 2.6: Wireframe illustration of the adopted multi-limbed floating-base system showing the main reference frames: the inertial $\mathcal{I}$, the base $\mathcal{B}$, and the centroidal $\mathcal{G}$ frames as well as the external contact forces.

multi-body inertias of our system

$$h_{com} = \begin{bmatrix} \mathbf{A_b}(\mathbf{q}) & \mathbf{A_j}(\mathbf{q}) \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_b \\ \dot{\mathbf{q}}_j \end{bmatrix}. \tag{2.14}$$

Rearranging (2.14), we obtain

$$\dot{\mathbf{q}}_\mathbf{b} = \mathbf{A_b}^{-1}\big(h_{com} - \mathbf{A_j}\dot{\mathbf{q}}_j\big), \tag{2.15}$$

where $\mathbf{q}_b = \big(r_{IB}, \mathit{\Phi}_{IB}^{zyx}\big) \in \mathbb{R}^6$ is the pose of the robot base with respect to the inertial frame.

As such, the robot dynamics are defined for motion planning purposes, with the state and input vectors being respectively

$$\boldsymbol{x}_r = \big(h_{com}, q_b, q_j\big) \in \mathbb{R}^{12+n_a}, \tag{2.16}$$

$$\boldsymbol{u} = \big(f_{c_1}, \dots, f_{c_{n_c}}, v_j\big) \in \mathbb{R}^{3n_c+n_a}, \tag{2.17}$$

where $\dot{q}_j = v_j$. Contact torques in this formulation are neglected, with the assumption that the contacts are formed at points rather than patches. The resulting system, as depicted in Fig. 2.6, has 3 joints per leg, and 4 joints for the arm. It also has 5 potential contact points, resulting in 28 state variables at 31 inputs.

For whole-body manipulation applications, we may augment the state of the object of interest that we want to manipulate within our system state, thus achieving a description of the robot-object dynamic coupling. For an object state $x_o = (q_0, v_0) \in \mathbb{R}^{2n_o}$, we obtain the augmented state $x = (x_r, x_0)$ for the full system flow-map $\dot{x} = f(x, u)$. The object dynamics are defined similarly to (2.12a),

$$\dot{x}_0 = \begin{bmatrix} v_0 \\ B_0^{-1}\big(- J_{c_o}^T f_{c_5} - b_0 \big) \end{bmatrix}, \tag{2.18}$$

where the term b_0 captures generalized forces depending on position and velocity. This sort of formulation requires knowledge of the object model and parameters, and requires that the state of the object is observable so as to be fed back to the MPC solver.

2.5 Constrained Numerical Optimization for Continuous-Time OCPs

In the following two sections, we will be addressing problems of the form

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{2.19a}$$

subject to

$$g(\boldsymbol{x}) = 0 \tag{2.19b}$$

$$h(\boldsymbol{x}) \geq 0, \tag{2.19c}$$

where $f(x)$ is our cost function of interest to be optimized, x is the state vector at hand, and $g(x)$ and $h(x)$ are the equality and equality constraints respectively.

Optimization routines, such as SQPs, solve such a class of problems iteratively, as depicted by Fig. 2.7, whereby necessary and sufficient optimality conditions of the first order and second order for constrained problems are ensured.

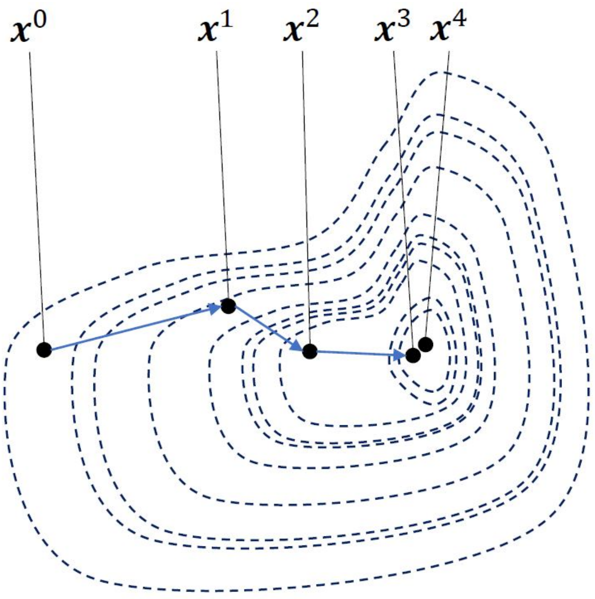


Figure 2.7: Qualitative example of iterative numerical optimization approach. The dashed lines represent the level curves of function f(x) [33].

Sequential-Quadratic Programming

The first class of algorithms we will address is SQP, whereby a sequence of primal, x, and dual, Lagrange multipliers, variables is generated that converges to a Karush-Kuhn-Tucker (KKT) triplet corresponding to a local minimizer. This method performs a quadratic approximation of the optimization program at each iteration k, and returns a solution in the space of search directions p_k, and Lagrange multipliers $\widetilde{\lambda_k}$ and $\widetilde{\mu_k}$. The approximated quadratic program (QP) around point $(p_k, \widetilde{\lambda_k}, \widetilde{\mu_k})$ will read:

$$\min_{p^k} \nabla_x f\left(x^k\right)^T p^k + \frac{1}{2} p^{k^T} H^k p^k \tag{2.20a}$$

subject to

$$\nabla_x g\left(x^k\right)^T p^k + g\left(x^k\right) = 0 \tag{2.20b}$$

$$\nabla_x h\left(x^k\right)^T p^k + h\left(x^k\right) \geq 0 \tag{2.20c}$$

Efficient solvers exist that tackle the solution of the resultant QP, each of them tailored to the specific characteristics of the problem at hand, two main encompassing families being interior-point methods and active set methods. The interior point method generates a sequence x_k^* in the interior of the feasibility region of the QP, and corresponds to the family of barrier methods. More specifically, interior-point strategies replace the non-smooth complimentarity conditions arising in the QP with a smooth nonlinear equation which gradually changes until the original conditions are recovered.

The QP approximation also requires the definition of a suitable Hessian matrix H^k, which approximates in a reasonable way the actual Hessian of the Lagrangian $\nabla_x^2 \mathcal{L}(p_k, \lambda_k, \mu_k)$, which may be too expensive to compute. Several methods can perform this approximation, including Broyden, Fletcher, Goldfarb, Shanno, more commonly known as BFGS, and Gauss Newton (GN). The latter is most useful when the cost function has the structure of a sum of squares. In this approach, the cost function is represented as the multiplication $F(x)^T F(x)$, with a suitably built function $F(x)$ that gathers the errors in state and input. The approximation then becomes $H^k = 2\nabla_x F\left(x^k\right) \nabla_x F\left(x^k\right)^T$, which is semi-positive definite. This fact contributes to the global convergence of the full algorithm. If needed, a small diagonal σI can be added to the resultant Hessian H^k to force positive definiteness in case numerical issues arise.

Another element that the QP requires is a suitable derivative computation method due to all the gradients involved in the approximation. Derivatives are typically computed analytically, using forward or central finite differences, or using automatic differentiation. Central finite differences typically provides a nice trade off between accuracy and computational speed.

After solving the QP, the following update rules are performed by the routine:

$$\begin{aligned} x_{k+1} &= x_k + t_k p_k \\ \lambda_{k+1} &= \lambda_k + t_k \varDelta\lambda_k \\ \mu_{k+1} &= \mu_k + t_k \varDelta\mu_k \end{aligned} \tag{2.21}$$

In order to define the step t_k taken by the algorithm, an auxiliary unconstrained back-tracking line search procedure is performed on the merit function of the original program. An l_1-norm merit function minimization of the form

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} T_1(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{p} \sigma_i \left|g_i(\boldsymbol{x})\right| + \sum_{i=1}^{q} \tau_i \max\left(0, -h_i(\boldsymbol{x})\right) \tag{2.22}$$

is often adopted, where σ_i and τ_i are tunable parameters that are automatically adjusted during the SQP routine. This method starts with a search line of length $\bar{t}$, and gradually decreases that value by a factor β until the condition $T_1\left(\boldsymbol{x}^k + t^k\boldsymbol{p}^{k*}\right) \leq T_1\left(\boldsymbol{x}^k\right) + t^k cD\left(T_1\left(\boldsymbol{x}^k\right), \boldsymbol{p}^{k*}\right)$ holds, which is called *Armijo Stopping Condition.* Other stopping conditions exist. Factor c represents the fraction of the desired improvement on the merit function over a linear approximation on the current point. D represents here the directional derivative. Even if that condition is never met, the algorithm considers a maximum number of backtracking iterations N_t^{max}. Other important parameters of the algorithm are the maximum number of overall SQP iterations N^{max}, the initial guess x_0, and the tolerances required to stop the algorithm: the gradient tolerance TOL_∇, the parameter advancement tolerance TOL_x, the cost function decrease tolerance TOL_f and the constraint satisfaction tolerance TOL_{constr}.

Using SQP, we can solve FHOCPs in continuous-time domain. FHOCPs have a similar but a more detailed description of the generic formulation in (2.19), as

$$\begin{aligned}
&\underset{\boldsymbol{u}(\cdot)}{\text{minimize}} && \phi(\boldsymbol{x}(t_f)) + \int_{t_s}^{t_f} l(\boldsymbol{x}(t), \boldsymbol{u}(t))dt && \text{(2.23a)}\\
&\text{subject to:} && \boldsymbol{x}(t_s) = \boldsymbol{x}_s && \text{(2.23b)}\\
& && \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) && \text{(2.23c)}\\
& && \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t) = \boldsymbol{0} && \text{(2.23d)}\\
& && \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, t) \geq \boldsymbol{0}, && \text{(2.23e)}
\end{aligned}$$

where $\boldsymbol{x}$, $\boldsymbol{u}$ are the state and control input respectively. SQP finds the minimum-cost (2.23a) trajectory under the system dynamics $\boldsymbol{f}$, the equality constraints $\boldsymbol{g}$, and the inequality constraints $\boldsymbol{h}$. In order to achieve MPC, we apply $\boldsymbol{u}(t_s)$, in practice, to the system in a receding horizon fashion to allow re-planning with a new measured state, $\boldsymbol{x}_s$.

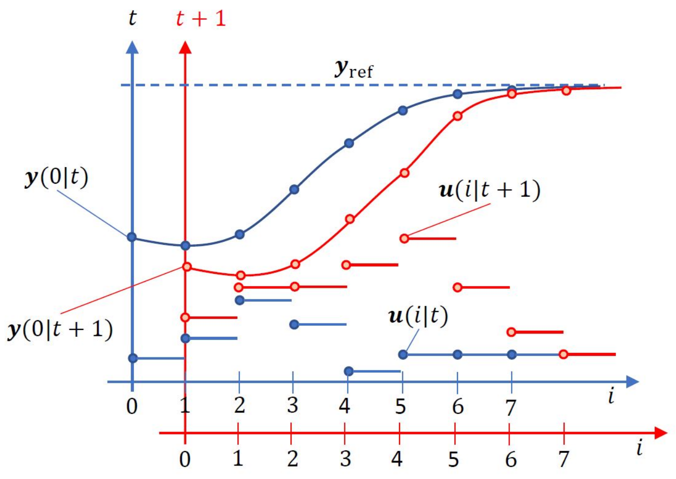


Figure 2.8: Sketch of receding horizon strategy for MPC [33]. At every time instance t, FHOCP is solved and only the first step of the emerging input solution at t+1 is commanded. New measurements are taken then FHOCP is solved all over again. y_{ref} is the reference trajectory, y(0|t) and y(0|t+1) are measurements at times t and t+1, u(i|t) and u(i|t+1) represent how the optimal solution changed from time t to t+1 at step i along the horizon.

Sequential-Linear Quadratic Programming

SLQ MPC is introduced in [36] as a receding horizon implementation of SLQ in a similar way as SQP MPC is a receding horizon implementation of SQP. It aims to solve a FHOCP in an unconstrained NMPC manner. Starting with the general nonlinear system

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.24}$$

where $\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ is assumed to be differentiable with respect to state and control input. The goal is to find a linear, time-varying feedback and feedforward controller of the form

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_{ff}(t) + \mathcal{K}(t)\mathbf{x}(\mathbf{t}), \tag{2.25}$$

with $\mathcal{K}(t)$ being the control gain matrix, and $u_{ff}(t)$ being the feedforward control action. By iteratively solving the FHOCP that minimizes the cost function

$$J = h(\mathbf{x}(t_f)) + \int_{t=0}^{t_f} l(\mathbf{x}(t), \mathbf{u}(t))dt, \tag{2.26}$$

we may be able to obtained the time-varying controller, where $l(\mathbf{x}, \mathbf{u})$ are the intermediate and terminal costs.

SLQ itself is described in [37]. Essentially, given the discrete system dynamics

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.27}$$

a cost function

$$J = h(\mathbf{x}(t_f)) + \sum_{t=0}^{t_f-1} l(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.28}$$

and an initially stable control law $\mathbf{u}(\mathbf{x}, t)$, SLQ first simulates the system dynamics with the control law from t(0) to $t_f - 1$, obtaining the trajectories of state and input τ: $x_n(0)$, $u_n(0)$, $x_n(1)$, $u_n(1)$, …, $x_n(t_f-1)$, $u_n(t_f-1)$, $x_n(t_f-1)$. It then linearizes the system dynamics along the obtained trajectory τ,

$$\delta\mathbf{x}(t+1) = \mathbf{A}(t)\delta\mathbf{x}(t) + \mathbf{B}(t)\delta\mathbf{u}(t), \tag{2.29}$$

where $\mathbf{A}(t) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}|_{\mathbf{x}(t),\mathbf{u}(t)}$ and $\mathbf{B}(t) = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}|_{\mathbf{x}(t),\mathbf{u}(t)}$ at every step along the obtained trajectory.

Next, it quadratizes the cost function along the trajectory τ as such

$$\begin{aligned}\tilde{J} \approx p(t) + \delta\mathbf{x}^T(t_f)\mathbf{p}(t_f) + \frac{1}{2}\delta\mathbf{x}^T(t_f)\mathbf{P}(t_f)\delta\mathbf{x}(t_f) + \sum_{t=0}^{t_f-1} q(t) + \delta\mathbf{x}^T\mathbf{q}(t) \\ + \delta\mathbf{u}^T\mathbf{r}(t) + \frac{1}{2}\delta\mathbf{x}^T\mathbf{Q}(t)\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{u}^T\mathbf{R}(t)\delta\mathbf{u},\end{aligned} \tag{2.30}$$

and backwards solves Riccati-like difference equations,

$$\begin{aligned}
&\mathbf{P}(t) = \mathbf{Q}(t) + \mathbf{A}^T(t)\mathbf{P}(t+1)\mathbf{A}(t) + \mathbf{K}^T(t)\mathbf{H}\mathbf{K}(t) + \mathbf{K}^T(t)\mathbf{G} + \mathbf{G}^T\mathbf{K}(t)\\
&\mathbf{p}(t) = \mathbf{q} + \mathbf{A}^T(t)\mathbf{p}(t+1) + \mathbf{K}^T(t)\mathbf{H}\mathbf{l}(t) + \mathbf{l}^T(t)\mathbf{g} + \mathbf{G}^T\mathbf{l}(t)\\
&\mathbf{H} = \mathbf{R}(t) + \mathbf{B}^T\mathbf{P}(t+1)\mathbf{B}(t)\\
&\mathbf{G} = \mathbf{B}^T(t)\mathbf{P}(t+1)\mathbf{A}(t)\\
&\mathbf{g} = \mathbf{r}(t) + \mathbf{B}^T(t)\mathbf{p}(t+1)\\
&\mathbf{K}(t) = -\mathbf{H}^{-1}\mathbf{G}\\
&\mathbf{l}(t) = -\mathbf{H}^{-1}\mathbf{g},
\end{aligned} \tag{2.31}$$

where $\mathbf{K}(t)$ is the feedback update and $\mathbf{l}(t)$ is the feedforward increment.

Finally, SLQ performs a line search to find the lower cost. Line search stops after a number of maximum line search steps is reached. Starting with an initial $\alpha = 1$, the line search first updates the control law as

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_n(t) + \alpha\mathbf{l}(t) + \mathbf{K}(t)(\mathbf{x}(t) - \mathbf{x}_n(t)), \tag{2.32}$$

then it forward simulates the system dynamics to obtain τ: $x_n(0)$, $u_n(0)$, $x_n(1)$, $u_n(1)$, $\ldots$, $x_n(t_f - 1)$, $u_n(t_f - 1)$, $x_n(t_f - 1)$. A new cost is then computed as $J = h(\mathbf{x}(t_f)) + \sum_{t=0}^{t_f-1} l(\mathbf{x}(t), \mathbf{u}(t))$ dt before scaling α by a constant factor α_d, $\alpha = \frac{\alpha}{\alpha_d}$.

This procedure is repeated until either of the stopping criteria:

1. maximum number of iterations is reached.
2. the procedure converged $\mathbf{l}(t) < l_t$

SLQ MPC entails running SLQ in an infinite loop, each iteration being triggered by a new state measurement and running until convergence. More technical details are covered in [36]. A cost function is chosen to have a quadratic shape since SLQ itself already computes a quadratic approximation, and it takes the form

$$J = \bar{x}(t_f)^T H \bar{x}(t_f) + \int_{t=0}^{t_f} \bar{x}(t)^T Q \bar{x}(t) + \bar{u}(t)^T R \bar{u}(t) + W(x, t) dt, \tag{2.33}$$

where $\bar{x}(t)$ and $\bar{u}(t)$ represent state and input deviations from the desired trajectory. H, Q, and R are the weighting matrices for the final, intermediate, and input costs. W(x,t) represents an intermediate way-point cost. One SLQ iteration has a linear complexity with the time horizon, allowing for efficient real-time applications.

Unconstrained-SLQ MPC has been reformulated for quadruped robots in [38] and extended to handle state-input and state-only equality constraints as well as inequality path constraints [39], through projections, penalty functions and barrier functions, respectively.

Chapter 3

Problem Formulation

This chapter poses the formulation of the OCPs we are trying to solve. Our goal is twofold:

1. A path following SLQ-MPC problem formulation that exercises direct control over the rate of arclength progression along a path, therefore adding a degree of freedom in time.
2. A holistic motion planning formulation for whole-body control of the robot while maximizing the probability of target visibility.

Due to the multi-objective nature of our problems, we opt for FHOCP formulations, which we solve in an MPC fashion. The optimal control problem is formulated in the continuous-time domain as seen before in (2.23), which we recall below,

$$\underset{\boldsymbol{u}(\cdot)}{\text{minimize}} \quad \phi(\boldsymbol{x}(t_f)) + \int_{t_s}^{t_f} l(\boldsymbol{x}(t), \boldsymbol{u}(t))dt \tag{2.23a}$$

$$\text{subject to:} \quad \boldsymbol{x}(t_s) = \boldsymbol{x}_s \tag{2.23b}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.23c}$$

$$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t) = \boldsymbol{0} \tag{2.23d}$$

$$\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, t) \geq \boldsymbol{0}, \tag{2.23e}$$

We solve both problems using SLQ solver [38] provided by the Optimal Control for Switched Systems (OCS2) toolbox [40]. We will consider that the visibility and orientation costs discussed in Sections 3.2.1 and 3.2.2 act on the robot end-effector, but they can generally act on any other frame. We assume that the camera that drives object detection and tracking is mounted on the end-effector.

3.1 Path Following SLQ MPC

As mentioned before, the main idea behind constrained path following in robotics is that we have a geometric path with no assigned speed, and we need to follow it optimally. In Fig. 3.1, we provide a sample path to be followed that is parametrized by the arclength

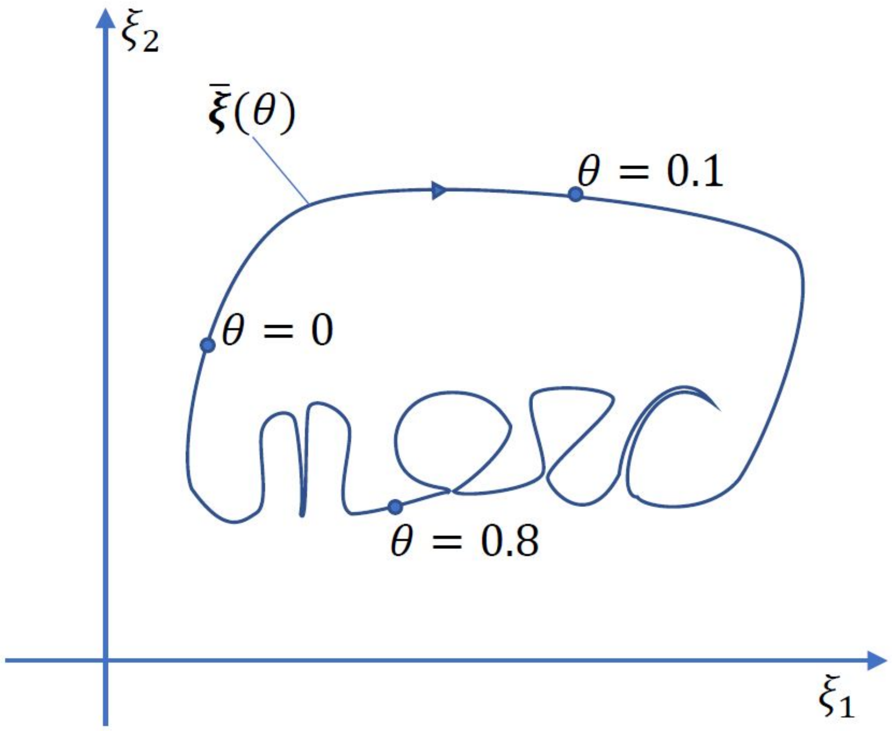


Figure 3.1: Example of a geometric 2D path to be followed [33]. The path is arclength parametrized, θ being the arc length parameter. The path starts at $\theta = 0$ and ends in $\theta = 1$, with ξ_1 and ξ_2 being the two dimensions of interest.

For our 3D mobile manipulator, we augment such an arclength parameter θ that parametrizes our 3D path within (2.10) to obtain the augmented state vector x_a with a size $n_{x_a} = 3 + n_j + 1 = 11$

$$\boldsymbol{x}_a = \begin{bmatrix} b_x & b_y & b_\psi & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & \theta \end{bmatrix}. \tag{3.2}$$

Setting the dynamics of our arclength parameter as

$$\dot{\theta} = \omega(t), \tag{3.3}$$

allows us to augment ω in the input vector (2.11) as

$$\boldsymbol{u} = \begin{bmatrix} v_b & \dot{\psi} & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 & \dot{q}_4 & \dot{q}_5 & \dot{q}_6 & \dot{q}_7 & \omega \end{bmatrix}, \tag{3.4}$$

resulting in $n_u = 10$ a which gives us full control over the rate of progression of the arclength parameter.

As such, we can command the progression along the path by constraining it to a desired value, ω_{des}, in a soft input constraint that penalizes the deviation as

$$\int_{t_s}^{t_f} \|\omega_{des} - \omega(t)\|_2^2 \, dt. \tag{3.5}$$

If we set ω_{des} to 1, we would be asking for a behaviour that resembles the behaviour of a trajectory tracking optimization. Otherwise, we can either ask for a faster, $w_{des} > 1$, a slower, $0 < w_{des} < 1$, or even a negative, $w_{des} < 0$ path progression. This value can be dynamically controlled.

The soft input constraint (3.5) controls the arclength parameter, but we still need to enforce the end-effector to follow the path at the emerging arclength value. We do that by adding a soft state cost that penalizes the difference between the current end-effector pose and the desired end-effector pose, the latter being specified by the arclength parametrization. At every step, a θ emerges based on the (3.3) and the pose along the path at the emerging θ will be the desired one. The current pose is directly extracted from the fed back measurements. That is, we penalize the error

$$\boldsymbol{x}(t) - \boldsymbol{x}_{des}(\theta). \tag{3.6}$$

The gradient of the error above can be easily computed on-demand. The derivative with respect to the state required for the linear approximation discussed in the Section 2.5 can be obtained as

$$\frac{\partial f}{\partial x} = -\frac{\partial x_{des}(\theta)}{\partial \theta} = -\nabla x_{des_\theta}, \tag{3.7}$$

which we compute using central finite differences as

$$-\nabla x_{des_\theta} = \frac{\big(x_{des}(\theta + \eta) - x_{des}(\theta - \eta)\big)}{2\eta}, \tag{3.8}$$

where $\eta = \sqrt{\epsilon}$ with ϵ being the machine precision.

It is common in literature to also specify constraints on the speeds $\dot{x}_e$ so as to minimize oscillations as in [2]. This can be easily added to our implementation. Moreover, in many applications, θ is constrained to be positive, so as to prevent backward progression along the path. In our case we do not specify that as we are not interested in only forward movements. We also only constrain the position of the end-effector, leaving its orientation unconstrained. One may keep the orientation fixed throughout the path, or may add an orientation parametrization. One may also constrain the orientation to stay locked onto a dynamically moving target, like for example the case of a videography application where the mobile manipulator moves along a certain path while keeping the target at the center of the video. We may achieve the latter for example by solving the following constrained problem,

$$\min_{q_0,q_1,q_2,q_3} f(q_0, q_1, q_2, q_3) = \begin{bmatrix} u_{des} \\ v_{des} \\ 1 \end{bmatrix} - \begin{bmatrix} u_{act} \\ v_{act} \\ 1 \end{bmatrix} \tag{3.9a}$$

subject to

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 0 \tag{3.9b}$$

where we are minimizing the difference between desired and actual camera pixel coordinates of the target. For example, if the camera has a 1920 $\times$ 1080 pixel resolution, one can set the desired coordinates to 960 and 540 respectively. The variables at play here are the components which need to be normalized. As such we are trying to find the quaternion that minimizes the error in pixel coordinates.

The relationship between the actual pixel coordinates and the quaternion can be obtained using the distortion-free projective transformation given by the pinhole camera model [41] as

$$sp = K[R \mid t]P_w, \tag{3.10}$$

where p represents the camera pixel coordinates, s is a scaling factor, P_w being the position of the target of interest, K is the matrix containing the camera intrinsic parameters, namely the focal lengths f_x and f_y, which are expressed in pixel units, and the principal point (c_x, c_y), that is usually close to the image center

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.11}$$

and R and t are the rotation matrix, as a function of the quaternion, and the translation that describe the the transformation from world to camera frame, also known as camera extrinsic parameters,

$$[R \mid t] = \begin{bmatrix} 2\left(q_0^2 + q_1^2\right) - 1 & 2\left(q_1 q_2 - q_0 q_3\right) & 2\left(q_1 q_3 + q_0 q_2\right) & X_{cam} \\ 2\left(q_1 q_2 + q_0 q_3\right) & 2\left(q_0^2 + q_2^2\right) - 1 & 2\left(q_2 q_3 - q_0 q_1\right) & Y_{cam} \\ 2\left(q_1 q_3 - q_0 q_2\right) & 2\left(q_2 q_3 + q_0 q_1\right) & 2\left(q_0^2 + q_3^2\right) - 1 & Z_{cam} \end{bmatrix}. \tag{3.12}$$

Assuming the camera is mounted on the end-effector, the minimization (3.9) depends on the emerging position of the camera along the horizon. At every step, we solve the optimization and dynamically set the orientation constraint using the resulting quaternion solution.

Collision Avoidance

In Fig. 3.2, we place an obstacle along the path, therefore expecting a collision while following the path. From perception, we can detect the position of the target, and therefore infer at which θ along the path our robot will collide with the obstacle. We can then place a relaxed log barrier at that theta value.

The relaxed log barrier, as the name suggests, is a relaxation of the typical logarithmic barrier function that is used in interior-point optimization routines to implement inequality constraints within the cost function and takes the form

$$\tau \sum_{i=1}^{n_{ineq}} \ln C_i \mathbf{x} + d_i, \tag{3.13}$$

where $C_i\mathbf{x}+d_i = 0$ represents one of the KKT conditions that represent an inequality condition $C_i\mathbf{x} + d_i$ in inequality-constrained QPs. As $\tau \to 0$, the optimal solution is approached over successive iterations. On the other hand, the logarithmic barrier function always tends to $+\infty$ as $C_i\mathbf{x} + d_i \to 0$ and gets sharper as $\tau \to 0$ as depicted in Fig. 3.3. Moreover,such a barrier is undefined outside the feasible space. This

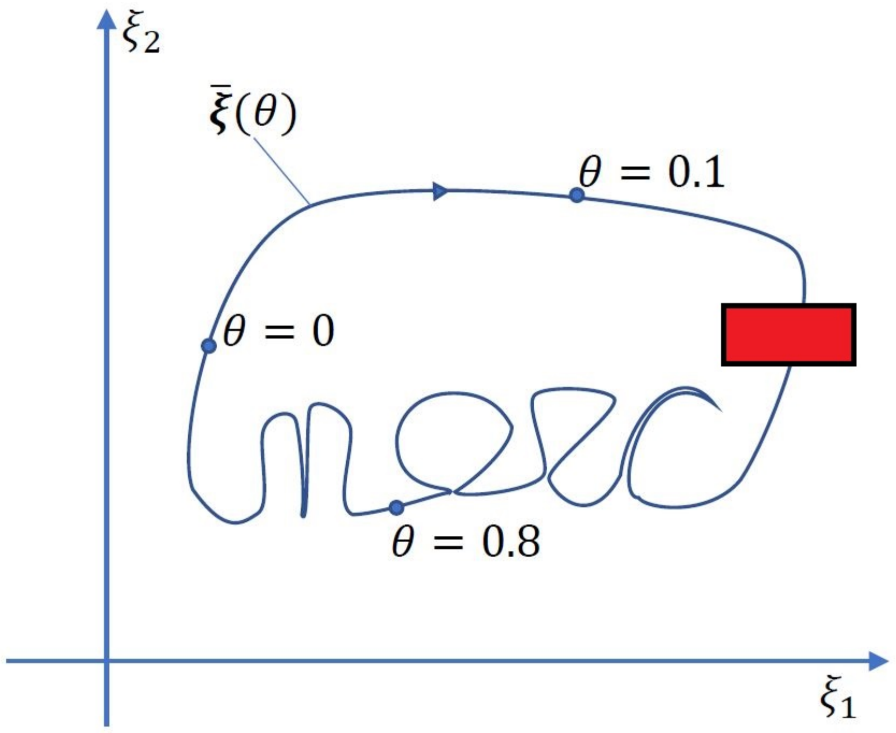


Figure 3.2: Same sketch as in Fig. 3.1 but with an obstacle (red box) blocking the path, anticipating a collision along the path.

makes it unsuitable for shooting methods as it causes ill-conditioned QP and LQ approximations.

The relaxed log barrier addresses those issues, making it more suitable for SLQ MPC. It switches between two different functions at the constraint boundary as

$$B(h) = \begin{cases} -\ln h, & \text{if } h \geq \delta \\ \beta(h; \delta), & \text{otherwise} \end{cases} \tag{3.14}$$

where h represents the inequality and δ represents the distance from the constraint boundary.

In the adopted SLQ MPC for this work, a quadratic extension for β is used

$$\beta(h; \delta) = \frac{1}{2}\left(\left(\frac{h - 2\delta}{\delta}^2 - 1\right)\right) - \ln \delta. \tag{3.15}$$

As such, a continuity of the barrier function is guaranteed, and the resulting function is depicted in Fig 3.4. As $\delta \to 0$, the standard logarithmic barrier is retrieved.

Therefore, the inequality based on $C_i\mathbf{x} + d_i = 0$ that we wrap with a relaxed log barrier that ensures collision avoidance has the shape

$$\theta(t) - \theta_{limit} = 0, \tag{3.16}$$

whose gradient with respect to the state $\frac{\partial f}{\partial x}$ is simply -1.

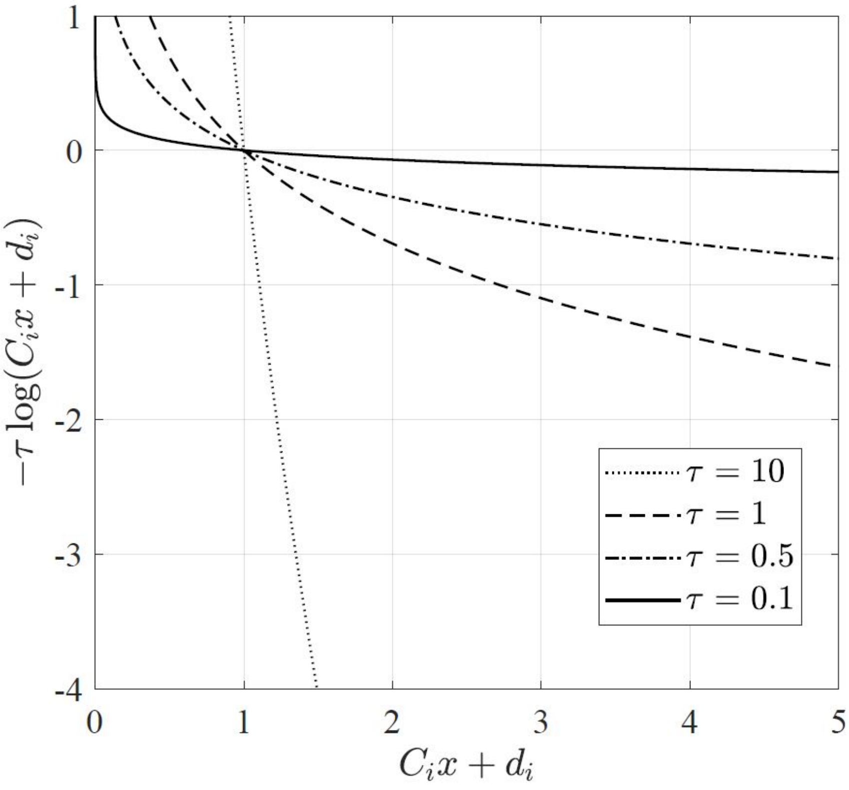


Figure 3.3: Logarithmic barrier function for the i_{th} inequality constraint for different values of τ [33].

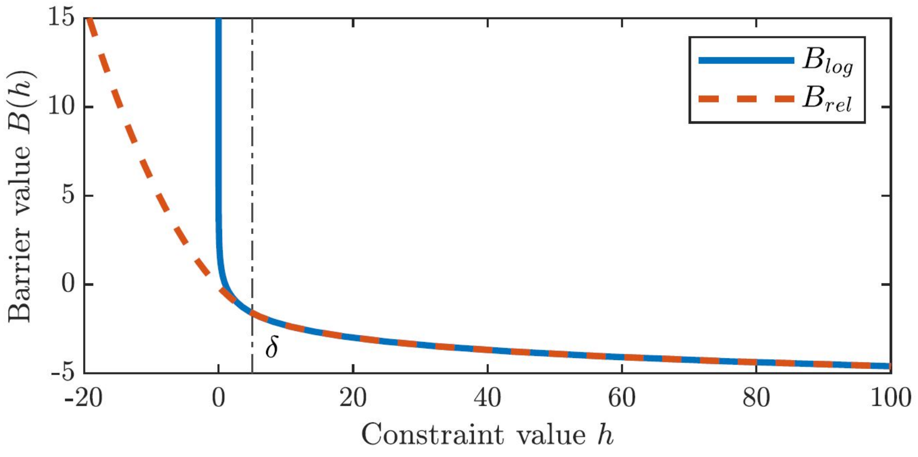


Figure 3.4: Relaxed log barrier B_{rel} comparison with the logarithmic barrier function B_{log} for $\delta = 5$ [39].

Input Costs, Joint Velocity Limits, Self-Collision Avoidance, and Final Cost

The main costs and constraints for achieving path following SLQ MPC formulation are now complete. The remaining costs and constraints are rather generic for almost every optimal control problem.

We penalize the quadratically form of the input costs as $u^T Ru$, while each joint velocity u_i nearing its operational limits is penalized with a relaxed barrier function similar to the previous subsection as seen in (3.17a). Distances between any pair of links/bodies in a defined set of pairs $\mathcal{P}$ of the robot can be extracted easily with Pinocchio, and self-collision avoidance can be achieved by wrapping those distances with relaxed log barriers to keep them larger than a minimum distance d_{min} as seen in (3.17b). Lastly, the final cost is simply a penalty to the error with respect to the final state.

$$\begin{cases} -u_{i_{max}} \leq u_i \leq u_{i_{max}} \quad \forall i \in \{1, \ldots, n_u\} & (3.17a) \\ d_i \geq d_{min} \quad \forall i \in \mathcal{P} & (3.17b) \end{cases}$$

3.2 Dynamic Occlusion Avoidance within Whole-Body Motion Planning

The material in this section is covered in a similar way to the previous one. We use the same FHOCP formulation (2.23), keeping in mind the dynamic modelling for ALMA presented . We will be introducing the two main costs, the visibility cost and the orientation cost, one at a time, then briefly discussing the remaining equality and inequality constraints.

3.2.1 The Reformulated Visibility Constraint

Typically, visibility constraints in control problems are enforced by restricting the pixel coordinates of the tracked/ detected object of interest within upper and lower bounds inside the current FOV as shown in (3.18)

$$\begin{bmatrix} u_{min} \\ v_{min} \end{bmatrix} \leq \begin{bmatrix} u_{act} \\ v_{act} \end{bmatrix} \leq \begin{bmatrix} u_{max} \\ v_{max} \end{bmatrix}. \tag{3.18}$$

The dynamics of the camera pixel coordinates can be described using the following equation,

$$\begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} u_k \\ v_k \end{bmatrix} + T_s I J_A \dot{q}. \tag{3.19}$$

where T_s represents the sampling period, $J_A \dot{q}$ represents the camera twist, and I is the interaction matrix,

$$I = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{u}{Z} & uv & -\left(1+u^2\right) & v \\ 0 & -\frac{1}{Z} & \frac{v}{Z} & 1+v^2 & -uv & -u \end{bmatrix}, \tag{3.20}$$

Z is the depth of the tracked object of interest, and IJ_A forms the Image Jacobian J_I. As such, this allows defining a cost over the function

$$f = \begin{bmatrix} u_{des} \\ v_{des} \end{bmatrix} - \begin{bmatrix} u_a \\ v_a \end{bmatrix}. \tag{3.21}$$

We reformulate those constraints as a maximization of the likelihood that the end-effector has a line of sight to the target. The likelihood function, however, is not time-separable, making it unsuitable for trajectory optimization methods. As such, rather than maximizing the likelihood of visibility, we maximize the log-likelihood.

$$\int_{t_s}^{t_f} \log \mathcal{F}(\boldsymbol{x}_{ee}(t))dt, \tag{3.22}$$

where $\mathcal{F}$ in (3.22) is the *shadow field* containing information about visibility and occlusion and $\boldsymbol{x}_{ee}$ is the 3D position of the end-effector frame of interest within that field. Since we formulated our problem as a minimization, we may augment (3.22) within (2.23a) as

$$l_{aug}(\boldsymbol{x}, \boldsymbol{u}) = l(\boldsymbol{x}, \boldsymbol{u}) - \log \mathcal{F}(\boldsymbol{x}_{ee}), \tag{3.23}$$

where the first term in (3.23) represents motion planning and system costs, and the second term represents the visibility cost. The gradient of such a cost can be obtained since the gradient of $\boldsymbol{x}_{ee}$ is the Jacobian of the end-effector, and the probabilistic *shadow field* is continuous and smooth, allowing for efficient on-demand trilinear gradient computation at every step. This gradient may also be computed using a Sobel filter approximation. Since the values returned by $\mathcal{F}(\boldsymbol{x}_{ee})$ can be zero, (3.22) is susceptible to extreme values due to the log function, making it unsuitable for shooting methods. Therefore, we use the relaxed log barrier penalty introduced in [39] and described in the previous section, as depicted in Fig. 3.5.

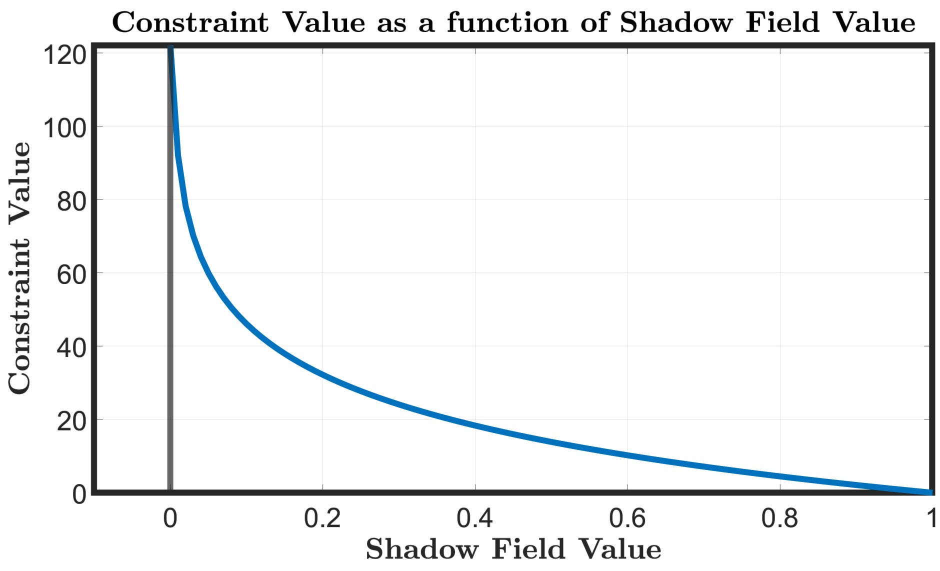


Figure 3.5: Relaxed log barrier wrapping of the shadow field cost.

3.2.2 The Orientation Constraint

The reformulated visibility constraint proposed in the previous subsection acts only on the end-effector position, leaving the end-effector orientation unconstrained. As a soft constraint, we add a cost that locks the end-effector onto the target, ensuring that the target remains within the camera's FOV, further enhancing the tracker's confidence about the target. Such a constraint is a function of the emerging solution from the planner along every step of the horizon. $\mathcal{V}$ in (3.24) represents the normalized directional vector from the emerging end-effector position $\mathcal{P}_{EE}$ to the light position (the target) $\mathcal{P}_L$. Using $\mathcal{V}$, we may extract the yaw and pitch angles in (3.25) and (3.26) respectively that orient the end-effector towards the target. The roll angle ψ remains free to choose.

$$\mathcal{V} = \mathcal{P}_L - \mathcal{P}_{EE} \tag{3.24}$$

$$\phi = \operatorname{atan2}(\mathcal{V}_1, \mathcal{V}_0) \tag{3.25}$$

$$\theta = -\operatorname{atan2}\left(\mathcal{V}_2, \sqrt{\mathcal{V}_0^2 + \mathcal{V}_1^2}\right). \tag{3.26}$$

We get the quaternion from those angles, which allows us to compute the normalized quaternion *error* between the desired orientation and the current end-effector orientation. This transformation allows us to safeguard against sudden tracker updates and faults by scaling the constraint's penalty by the factor

$$\gamma = \frac{\max\left(\alpha, \log\left(\frac{\iota * error^{\mathsf{c}}}{1 - error^{\mathsf{c}}}\right)\right)}{\beta}, \tag{3.27}$$

where $0 < \alpha < 1$, $\beta \neq 0$, and $\iota > 0$ are tuning parameters and $error^{\mathsf{c}}$ is the complement of the normalized quaternion *error*. In this manner, the end-effector gradually and progressively locks onto the target as the quaternion error decreases. We add the cost corresponding to this *error* minimization to l_{aug}, constituting the second term of the resulting total cost function as shown in (3.28)

$$l_{total}(\boldsymbol{x}, \boldsymbol{u}) = l_{aug}(\boldsymbol{x}, \boldsymbol{u}) + l_o(\boldsymbol{x_{ee}}). \tag{3.28}$$

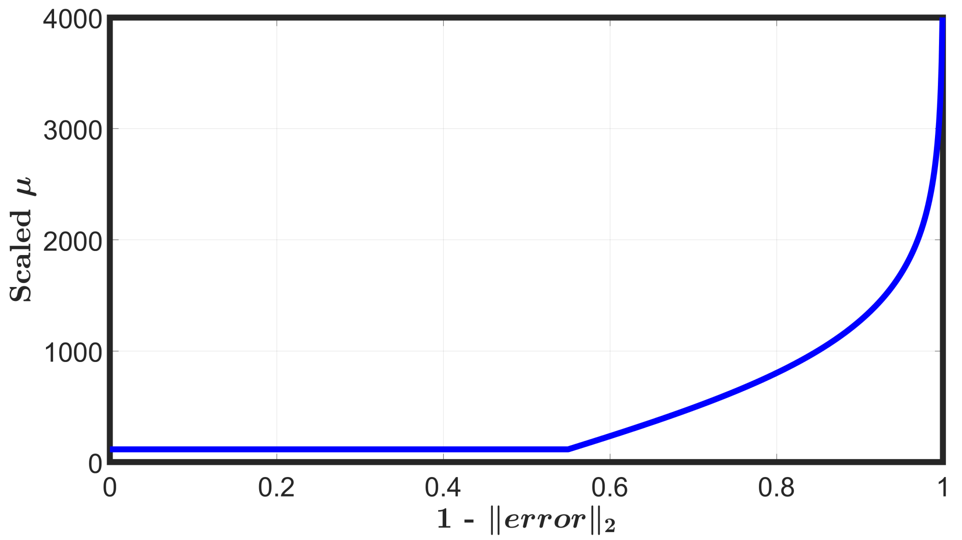


Figure 3.6: Logit scaling of the orientation penalty μ as a function of the complement of the normalized quaternion error. In simpler words, the desired behaviour is low penalty when the orientation error is large, and large penalty when the orientation error is low. This results in the end-effector slowly and progressively locking its orientation towards the target of interest. This safeguards the end-effector against cases of sudden tracker updates or any unforeseen change in the orientation.

3.2.3 Equality and Inequality Constraints

This subsection includes a brief overview of the equality and inequality constraints applied to ALMA. For more details, the reader is referred to [35].

Equality constraints for ALMA are defined for potential contact points, which may be either open or closed. Contacts are split in sets. They can be either closed, $\mathcal{C}$, at the feet, $\mathcal{F}$, or at the arm, $\mathcal{A}$, the latter being a singleton in this case. Since the robot is performing gaits during its motion, the contacts at the limbs are dynamically changing according to a predefined mode-schedule consisting of a mode-sequence and a set of switching times. We refer by $\mathcal{C}_j$ to a fixed mode instance starting at time s_j^0 and ending at s_j^f. The corresponding state-input equality constraints (2.23d) $g_j(x,u,t)$ $\forall t \in [s_j^0, s_j^f]$ and $\forall i \in \{1,\dots,n_c\}$

$$\begin{cases} v_{c_i} = 0 & \text{if } c_i \in \mathcal{C}_j \wedge c_i \in \mathcal{F} & (3.29a) \\ v_{c_i} \cdot \hat{n} = v^*(t) & \text{if } c_i \in \bar{\mathcal{C}}_j \wedge c_i \in \mathcal{F} & (3.29b) \\ v_{c_i} - J_{c_o} v_o = 0 & \text{if } c_i \in \mathcal{C}_j \wedge c_i \in \mathcal{A} & (3.29c) \\ f_{c_i} = 0 & \text{if } c_i \in \bar{\mathcal{C}}_j & (3.29d) \end{cases}$$

where c_i is the $i_=th$ contact point, v_{c_i} is the absolute linear velocity at that point expressed in inertial frame, $\hat{n}$ is the surface normal, $v^*(t)$ is a reference trajectory along $\hat{n}$, J_{c_o}, and f_{c_i} is the force at c_i.

Equality (3.29a) is a no-slip condition between ground and a stance leg at the point of contact, equality (3.29b) requires that swinging legs should track a reference trajectory along the surface normal, (3.29c) specifies the grasping condition of the grasped object at the gripping point, and the last equality (3.29d) assigns no force at the open contacts.

Inequality constraints, on the other hand, are used to ensure a that the generated motions and forces respect the system's operational limits as described below,

$$\begin{cases} -v_{j_{max}} \leq v_j^{arm} \leq v_{j_{max}} & & (3.30a) \\ -\tau_{max} \leq J_{c_i}^{a^T} f_{c_i} + g^a \leq \tau_{max} & \text{if } c_i \in \mathcal{C}_j \wedge c_i \in \mathcal{A} & (3.30b) \\ -\mu_s f_{c_i}^z - \sqrt{f_{c_i}^{x^2} + f_{c_i}^{y^2} + \epsilon^2} \geq 0 & \text{if } c_i \in \mathcal{C}_j \wedge c_i \in \mathcal{F} & (3.30c) \end{cases}$$

where J_a and J_g are the arm Jacobian and generalized gravitational torques, respectively, μ_s is a friction coefficient.

The inequality constraints ground the arm joint velocities and torques within limits, ground the feet contact forces within the friction cone with a coefficient μ_s, while $\epsilon \neq 0$ is a parameter to smoothen the constraint.

Chapter 4

Shadow Field

In Section 3.2.1, we assumed to have a field $\mathcal{F}$ that contains knowledge about likelihood of visibility across the space of interest as a function of the end-effector $\log \mathcal{F}(\boldsymbol{x}_{ee}(t))$. We also assumed it to be differentiable, continuous, and smooth enough to be used in our gradient-based optimal control formulation. In Section 1.2, we stated that commonly used methods to represent the notion of visibility and obstruction in the literature are quiet limited. Those methods are also not applicable for our formulation, due to being inefficient and/or non-differentiable or non-smooth. Moreover, most methods provide hard discrete shadows, meaning 1 for when the target is visible, 0 when it's invisible. As such, we propose our own DP-based probabilistic algorithm that returns the likelihood of point light visibility from any position in the space of interest. It then can be used along the MPC horizon to provide predictive power as can be seen in Fig. 4.1.

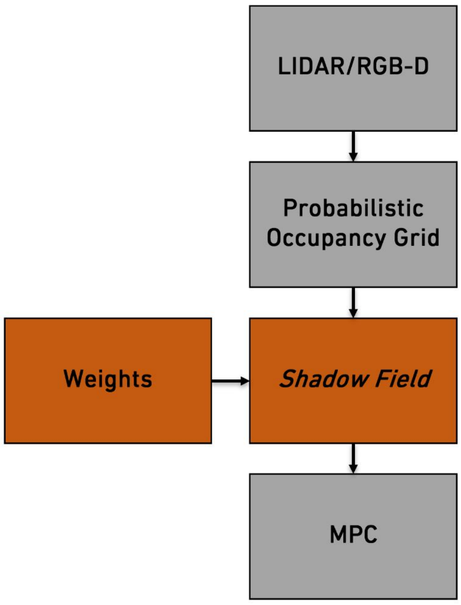


Figure 4.1: *Shadow field* mapping control and architecture. The *shadow field* is constructed using a probabilistic occupancy grid, which itself is constructed using LIDAR or RGB-D data. The field itself require a precomputation of weights also.

4.1 The Visibility Problem

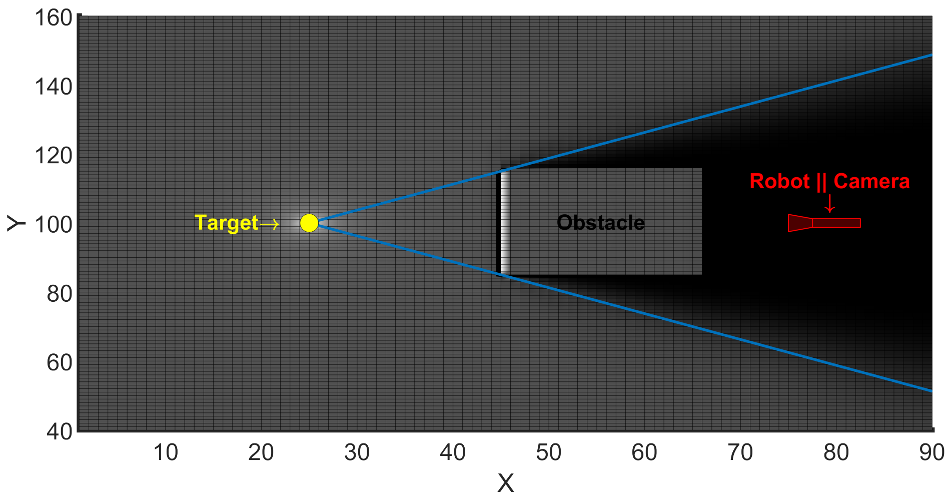


Figure 4.2: The visibility problem showing an obstacle occluding a point light source. The hard shadow cast by the obstacle is defined by the blue lines.

The visibility problem at hand is illustrated in Fig. 4.2. The tracked target of interest is considered to be a point light source. The setting is a discrete grid with a defined resolution. We update the *shadow field* corresponding to the grid using information contained in the occupancy grid. The basic idea is to propagate information about visibility as probabilities starting from the light position, moving outward in every direction one layer at a time, and updating the next layer of voxels based on the previous layer. The result is a smooth and differentiable field containing information about visibility for every voxel. This is equivalent to performing ray-casting from the light source to every voxel, but in a probabilistic DP manner.

4.2 *Shadow Field* Algorithms

2D *Shadow Field*

We first introduce the 2D case, which is a collapsed simpler version of the 3D case, then extend the reasoning to the 3D one. In a 2D grid, pixels of the prior layer L_{i-1} attenuate light based on their own probability of being occupied, consequently affecting neighboring pixels of the upcoming layer L_i. Each pixel in L_i is affected by exactly 2 neighbouring pixels in L_{i-1}. The contribution of each of those 2 pixels to light attenuation depends on the position of the queried pixel relative to the light source. The resulting mappings between the queried pixel and the contributions of the two neighboring pixels are constant due to the fact that they only depend on query point and grid resolution. This allows computing them only once during initialization to be cached and used at a later stage.

In the Fig. 4.3, the first quadrant of an XY plane is centered at the light position where three samples queried black pixels and their corresponding neighboring red, and blue pixels are displayed. For each black pixel, two black vectors passing through $(i+1, j)$ and $(i, j+1)$ and a green one passing through (i, j) are highlighted, referred to as $\vec{v}_x$, $\vec{v}_y$, $\vec{v}_m$ respectively. The two angles $\widehat{\vec{v}_m\vec{v}_x}$ and $\widehat{\vec{v}_y\vec{v}_m}$, and their sum $\widehat{\vec{v}_y\vec{v}_x}$, are computed using trigonometry.

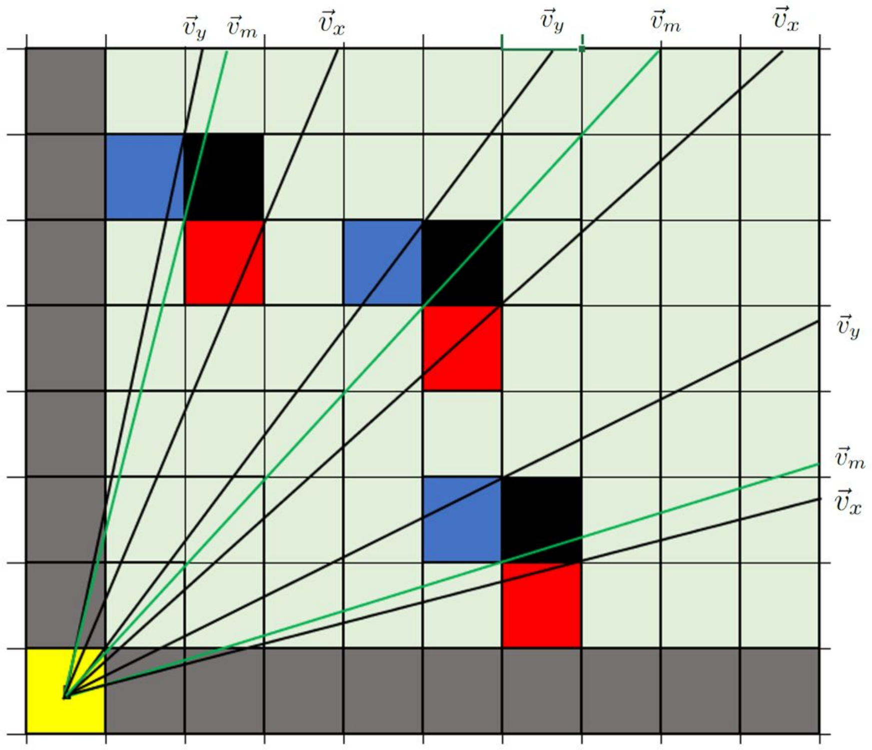


Figure 4.3: First quadrant of the XY plane showing the queried black pixel of interest in layer L_i and its two neighbouring red and blue pixels in layer L_{i-1}. v_m, v_x, and v_y are also visualized passing through points (i, j), $(i + 1, j)$ and $(i, j + 1)$ respectively.

More specifically, angles $\widehat{\vec{v}_x\vec{x}}$, $\widehat{\vec{v}_m\vec{x}}$, and $\widehat{\vec{v}_y\vec{x}}$, where $\vec{x}$ is the x-axis, are computed using

$$\begin{cases} \widehat{\vec{v}_x\vec{x}} = \arctan\left(\dfrac{j}{i+1}\right) & (4.1a) \\ \widehat{\vec{v}_m\vec{x}} = \arctan\left(\dfrac{j}{i}\right) & (4.1b) \\ \widehat{\vec{v}_y\vec{x}} = \arctan\left(\dfrac{j+1}{i}\right) & (4.1c) \end{cases}$$

from which we calculate

$$\begin{cases} \widehat{\vec{v}_m\vec{v}_x} = \widehat{\vec{v}_m\vec{x}} - \widehat{\vec{v}_x\vec{x}} & (4.2a) \\ \widehat{\vec{v}_y\vec{v}_m} = \widehat{\vec{v}_y\vec{x}} - \widehat{\vec{v}_m\vec{x}} & (4.2b) \\ \widehat{\vec{v}_y\vec{v}_x} = \widehat{\vec{v}_y\vec{x}} - \widehat{\vec{v}_x\vec{x}}. & (4.2c) \end{cases}$$

The contributions of every red and blue pixel are set equal to the ratios of $\widehat{\vec{v}_m\vec{v}_x}$ to $\widehat{\vec{v}_y\vec{v}_x}$ and $\widehat{\vec{v}_y\vec{v}_m}$ to $\widehat{\vec{v}_y\vec{v}_x}$ respectively. Referring to Fig. 4.4, one may notice that as the black pixel moves further along x relative to y or vice versa, one angle progressively dominates the other. More intuitively, this means that as the black pixel moves downwards towards the x-axis, the light coming from the source will be progressively affected by the blue pixel and regressively affected by the red one. This justifies the assignment of weights to the red and blue pixels, or in other words, the mathematical equations (4.2) and (4.2) provide an accurate description of the observed behaviour.

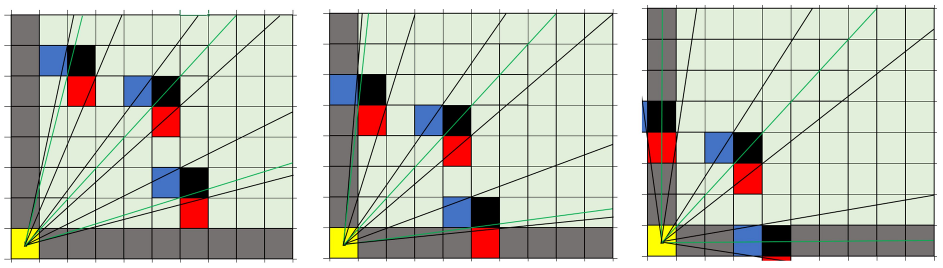


Figure 4.4: Illustration depicting the evolution of the contribution of red and blue pixels to blocking the black pixel's line of sight with the light source.

After updating the weights of the neighbouring red and blue pixels for every black pixel in the space of interest, the resulting distributions can be seen in Fig. 4.5.

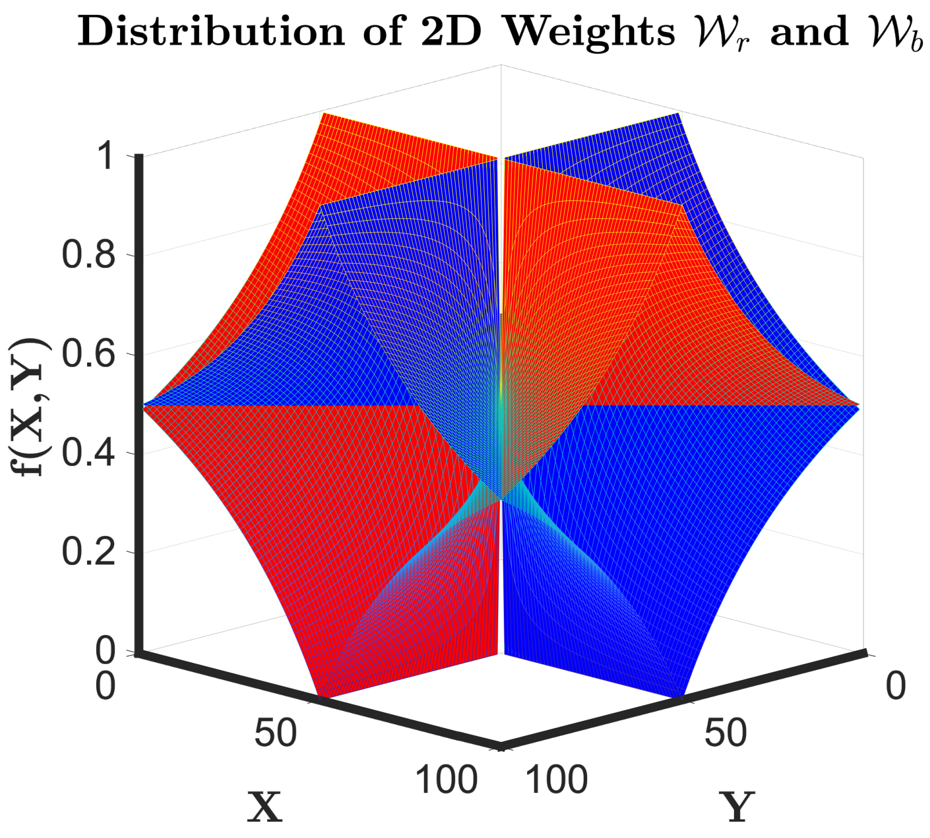


Figure 4.5: Shadow field 2D weights distributions as a tangent and its complement. Starting from the x-axis, the blue has maximum weights while the red has a weight of zero. As we sweep towards the diagonal, the two weights tend to be equal. Beyond that point, red will tend to 1 as we near the y-axis and vice versa for blue, which goes to zero. The two red and blue distributions are stored in matrices $\mathcal{W}_r$ and $\mathcal{W}_b$.

In Fig. 4.6, we compare hard shadow versus our soft shadow values for Fig. 4.2 scene on the vertical line $X = 70$. The comparison of these two techniques highlights the extent and accuracy of our soft shadow approach. Owing to its probabilistic nature and in contrast to a hard shadow technique, our *shadow field* provides a smooth representation of visibility. This smoothness plays an essential role for applications where the field's gradient is required, such as our MPC formulation.

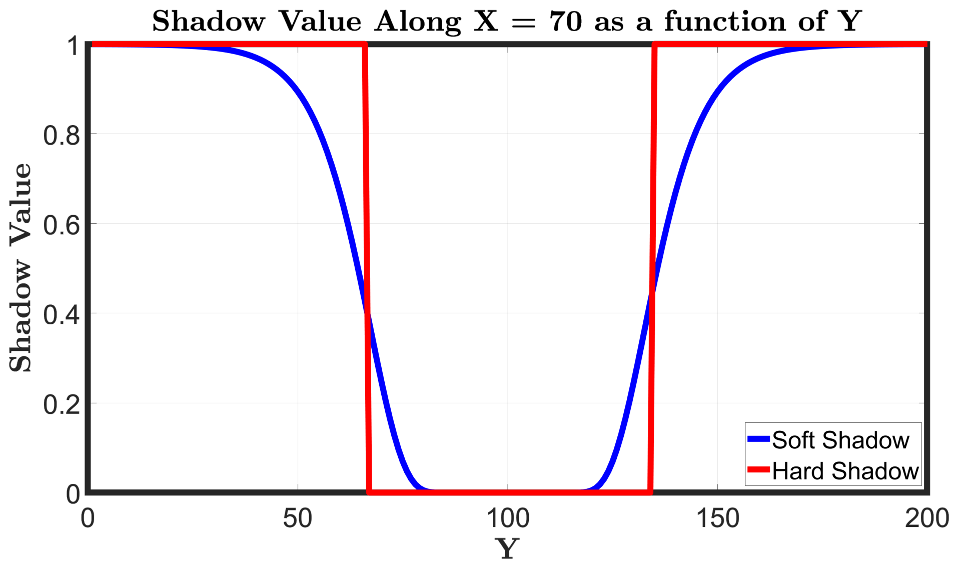


Figure 4.6: Values of our soft *shadow field* approximation against hard shadow values produced by the obstacle in Fig. 4.2 along X = 70.

Algorithm 1 *Shadow Field* Update Routine in 2D

```
Inputs: $\mathcal{L} \leftarrow$ grid sized $(x, y)$ valued 1.0
        $\mathcal{O} \leftarrow$ probabilistic occupancy grid of the map
        $(l_x, l_y) \leftarrow$ light indices
        $(x_p, y_p) \leftarrow$ # of indices till upper boundary of $\mathcal{L}$
        $(x_n, y_n) \leftarrow$ # of indices till lower boundary of $\mathcal{L}$
        $\mathcal{P}_{\mathcal{O}_{thresh}} \leftarrow$ confidence level for occupancy
        $\mathcal{W}_r, \mathcal{W}_b \leftarrow$ cached weights
Output: $\mathcal{F} \leftarrow$ 2D probabilistic field
1:  procedure UPDATESHADOWFIELD()                    ▷ for first quadrant only
2:      Initialize $\mathcal{F}$ to $\mathcal{L}$
3:      for $x_i = 0$ to $x_p$ , $y_i = 0$ to $y_p$, do
4:          $x_k = l_x + x_i$
5:          $y_k = l_y + y_i$
6:          $\mathcal{P}_{\mathcal{O}} \leftarrow \mathcal{O}(x_k, y_k)$
7:          if $\mathcal{P}_{\mathcal{O}} > P_{\mathcal{O}_{thresh}}$ then
8:              $\mathcal{F}(x_k, y_k) \leftarrow (1 - P_{\mathcal{O}})$
9:          else
10:             $\mathcal{C}_r \leftarrow \mathcal{W}_r(x_i, y_i) * \mathcal{F}(x_{k-1}, y_k)$
                $\mathcal{C}_b \leftarrow \mathcal{W}_b(x_i, y_i) * \mathcal{F}(x_k, y_{k-1})$
                $\mathcal{F}(x_k, y_k) \leftarrow \mathcal{C}_r + \mathcal{C}_b$
11:     return $F$
```

We summarize the procedure for updating a 2D *shadow field* in 1. The update procedure is centered around the light position, and iterates in 4 quadrants around it. For brevity, we only discuss the first quadrant update formula, since the other three quadrants differ only in sign changes to iterate accordingly around the light position. The weights are always relative to the light position, while all other operations are relative to the center (0,0). Alg. 1 takes as input a probabilistic occupancy grid $\mathcal{O}$, the indices corresponding to the light positions l_x and l_y, the cached weights $\mathcal{W}_r$ and $\mathcal{W}_b$, and a tunable confidence level for the probability of being occupied $\mathcal{P}_{\mathcal{O}_{thresh}}$. Alg. 1 outputs the *shadow field*. Alg. 1 updates all pixels in the field one quadrant at a time, starting from the light source position. For every pixel in level, L_i, if its occupancy probability $\mathcal{P}_{\mathcal{O}}$ is higher than the threshold $\mathcal{P}_{\mathcal{O}_{thresh}}$, then assign to it a value of 1 - $\mathcal{P}_{\mathcal{O}}$, otherwise update it based on contributions of neighbouring red and blue pixels from L_{i-1}: $\mathcal{C}_r$ and $\mathcal{C}_b$.

We display in Fig. 4.7 three sample meshes representing the *shadow field* of a 2D environment grid of size 100×100 at two different views as we vary the light position dynamically. The generated wavy *shadow field* meshes in the side-views are characterized by the same smoothness as the soft shadow curvature shown in Fig. 4.6, meaning both umbras and penumbras are produced for each occluding obstacle. The umbras are characterized by the flat shadow gradient, or by the pitch-black regions, where the likelihood of having a line of sight there with the target is slim-to-none. The penumbras are the shades of gray at the outer regions of the shadows and these are the smooth and curvy regions in the mesh. The algorithm's DP nature allows us to run it at more than 10kHz for a 2D scenario in C++.

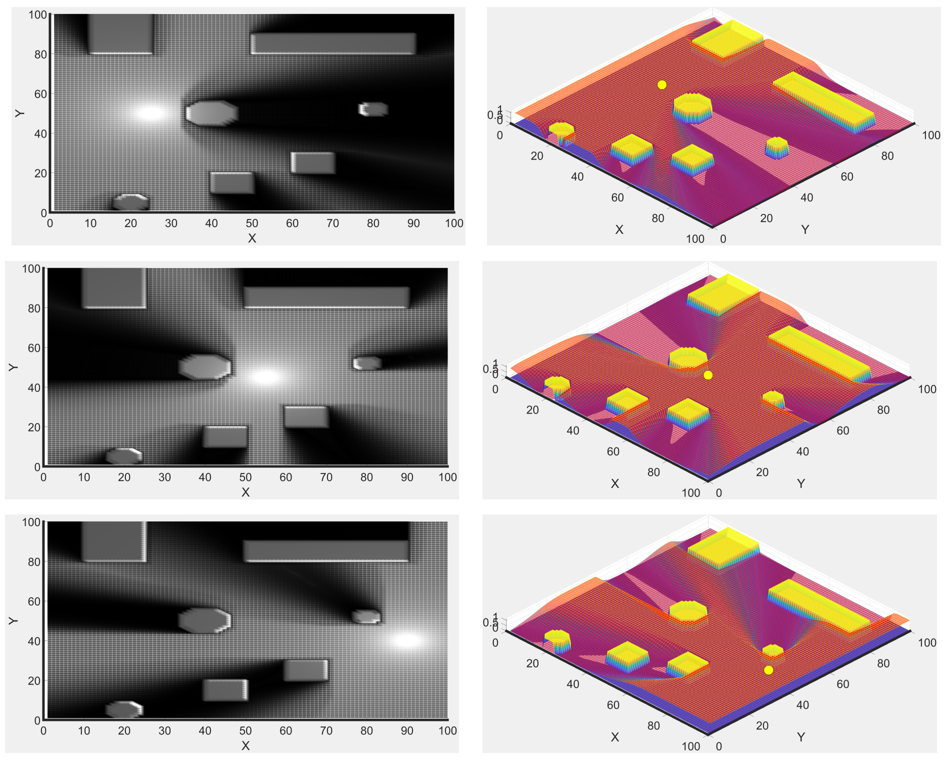


Figure 4.7: Three pairs of one top-view and its corresponding side-view of a 2D *shadow field* that results from obstacles (yellow face color) occluding a point light source. The environment is identical in all illustrations, the only difference being the light position.

3D *Shadow Field*

Since the 3D scenario is an extension of the 2D case, where in addition to the XY plane one needs to consider the XZ and YZ planes, the same reasoning can be extended from 2D to 3D. Each queried voxel (3D pixel) in L_i is affected by exactly 3 neighbouring ones in L_{i-1}. An example of a queried 3D voxel with its three neighbours is shown in the first illustration in Fig. 4.8 as black, red, blue, and gold respectively, whereas the light position is the yellow one. Also, four black vectors are shown to be extending from the light position to locations (x_i, y_i, z_i), (x_{i+1}, y_i, z_i), (x_i, y_{i+1}, z_i), and (x_i, y_i, z_{i+1}), which represent the queried voxel and its three relevant neighbours, as seen on the left in Fig. 4.10. Those vectors are referred to $\vec{v}_m$, $\vec{v}_x$, $\vec{v}_y$, and $\vec{v}_z$ in Alg. 2. The normals to the planes formed by vectors $\vec{v}_x$ and $\vec{v}_y$, $\vec{v}_x$ and $\vec{v}_z$, and $\vec{v}_y$ and $\vec{v}_z$ are then computed. The planes themselves can be seen on the right in Fig. 4.10. Using those normals, the angles between vector $\vec{v}_m$ and the three aforementioned planes as well as their sum may be obtained. In the final step, the respective weights distributions for every voxel are calculated and stored in the mappings $\mathcal{W}_r$, $\mathcal{W}_b$, and $\mathcal{W}_g$.

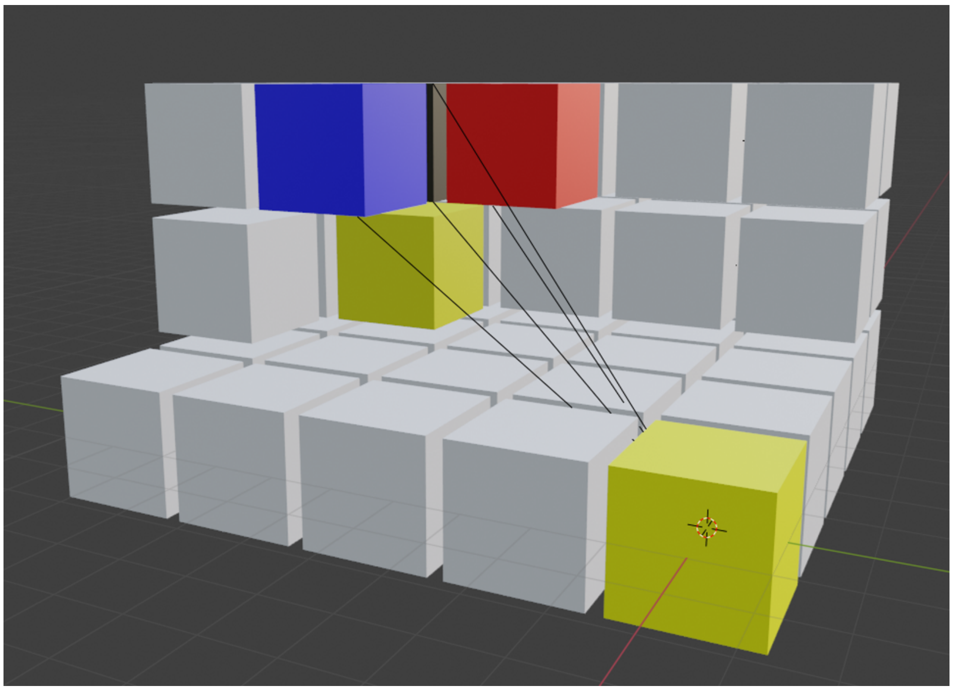


Figure 4.8: Illustration depicting the queried black voxel of interest and its three red, blue, and gold neighbours. Voxel corresponding to light position is yellow.

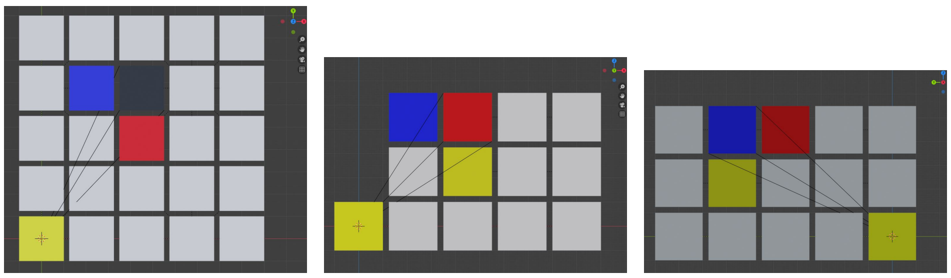


Figure 4.9: XY, XZ, and YZ views of Fig. 4.8.

Weights Algorithm

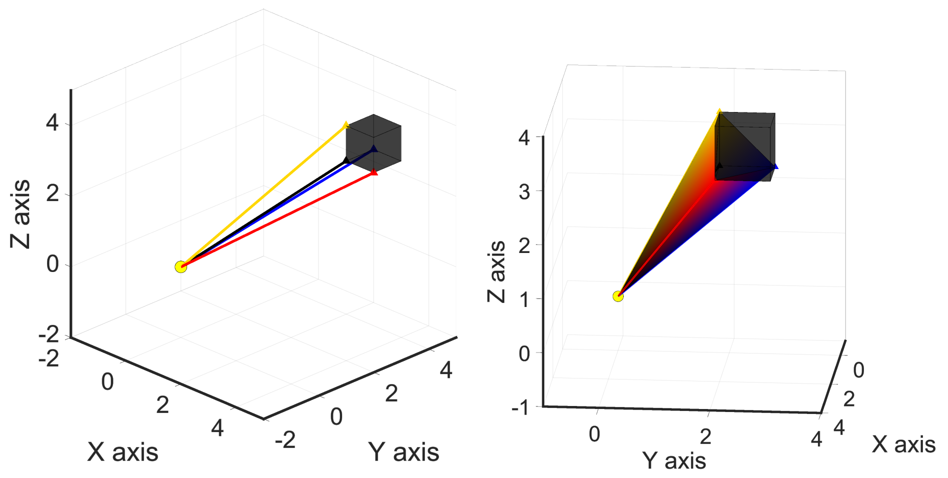


Figure 4.10: A queried 3D cube and its relevant vectors and planes formed by those vectors.

As mentioned in the previous subsection, for the 3D case we no longer consider only angles between two vectors, but rather the angles between the central black vector $\vec{v}_m$ shown on the left in Fig. 4.10 and the three planes formed by vectors $\vec{v}_x$, $\vec{v}_y$, and $\vec{v}_z$. Those angles may be computed by first inferring the normals of the aforementioned planes, $\vec{n}_{xy}$, $\vec{n}_{xz}$, $\vec{n}_{yz}$. Then as in the case of 2D, we update the three weights distributions accordingly. This procedure is summarized in 2.

Algorithm 2 *Shadow Field* Weights Routine in 3D

```
1: procedure INITSHADOWFIELDWEIGHTS()
2:     $\mathcal{T} \leftarrow$ empty volume of size$(x, y, z)$
3:     Initialize $\mathcal{W}_r$, $\mathcal{W}_b$, and $\mathcal{W}_g$ to $\mathcal{T}$
4:     for $x_i$ in $x$, $y_i$ in $y$, $z_i$ in $z$, $(x, y, z) \in \mathcal{T}$, do
5:         $\vec{v}_m \leftarrow (x_i, y_i, z_i)$
           $\vec{v}_x \leftarrow (x_{i+1}, y_i, z_i)$
           $\vec{v}_y \leftarrow (x_i, y_{i+1}, z_i)$
           $\vec{v}_z \leftarrow (x_i, y_i, z_{i+1})$
6:         $\vec{n}_{xy} \leftarrow \vec{v}_y \times \vec{v}_x$
           $n_{xz} \leftarrow \vec{v}_x \times \vec{v}_z$
           $n_{yz} \leftarrow \vec{v}_z \times \vec{v}_y$
7:         $a_{xy} \leftarrow \arcsin\left(\frac{\vec{v}_m \cdot \vec{n}_{xy}}{\|\vec{v}_m\| \cdot \|\vec{n}_{xy}\|}\right)$
           $a_{xz} \leftarrow \arcsin\left(\frac{\vec{v}_m \cdot \vec{n}_{xz}}{\|\vec{v}_m\| \cdot \|\vec{n}_{xz}\|}\right)$
           $a_{yz} \leftarrow \arcsin\left(\frac{\vec{v}_m \cdot \vec{n}_{yz}}{\|\vec{v}_m\| \cdot \|\vec{n}_{yz}\|}\right)$
           $a_{sum} \leftarrow (a_{xy} + a_{xz} + a_{yz})$
8:         $\mathcal{W}_r(x_i, y_i, z_i) \leftarrow \frac{a_{yz}}{a_{sum}}$
           $\mathcal{W}_b(x_i, y_i, z_i) \leftarrow \frac{a_{xz}}{a_{sum}}$
           $\mathcal{W}_g(x_i, y_i, z_i) \leftarrow \frac{a_{xy}}{a_{sum}}$
9:     return $\mathcal{W}_r$, $\mathcal{W}_b$, $\mathcal{W}_g$
```

Main Update Routine

Shadow field constitutes a local subset of the global map and is centered around the light source, with a maximum number of indices separating between the light source index and upper and lower bounds of the *shadow field*, (x_p, y_p, z_p) and (x_n, y_n, z_n) respectively, as can be seen in Fig. 4.11. The logic for updating the 3D *shadow field* is very similar to that of the 2D case. Alg. 3 takes those numbers as input, as well as the probabilistic occupancy grid $\mathcal{O}$, the indices corresponding to the global and local light positions $(l_{l_x}, l_{l_y}, l_{l_z})$ and $(l_{g_x}, l_{g_y}, l_{g_z})$, the cached weights $\mathcal{W}_r$, $\mathcal{W}_b$, $\mathcal{W}_g$, and a tunable confidence level for the probability of being occupied $\mathcal{P}_{\mathcal{O}_{thresh}}$. Alg. 3 outputs the *shadow field*. Alg. 3 updates all cells in the field one quadrant at a time, starting from the light source position. For every voxel in level, L_i, if its occupancy probability $\mathcal{P}_{\mathcal{O}}$ is higher than the threshold $\mathcal{P}_{\mathcal{O}_{thresh}}$, then assign to it a value of 1 - $\mathcal{P}_{\mathcal{O}}$, otherwise update it based on contributions of neighbouring red, blue, and golden voxels from L_{i-1}: $\mathcal{C}_r$, $\mathcal{C}_b$, and $\mathcal{C}_g$. For brevity, we also only discuss the first quadrant update formula, since each of the other quadrants requires only few sign changes in the iteration logic.

Algorithm 3 *Shadow Field* Update Routine in 3D

```
Inputs: $\mathcal{L} \leftarrow$ volume $(x_{local}, y_{local}, z_{local})$ valued 1.0
        $\mathcal{O} \leftarrow$ probabilistic occupancy grid of the global map
        $(l_{g_x}, l_{g_y}, l_{g_z}) \leftarrow$ light indices in global map
        $(l_{l_x}, l_{l_y}, l_{l_z}) \leftarrow$ light indices in local map
        $(x_p, y_p, z_p) \leftarrow$ # of indices till upper boundary of $\mathcal{L}$
        $(x_n, y_n, z_n) \leftarrow$ # of indices till lower boundary of $\mathcal{L}$
        $\mathcal{P}_{\mathcal{O}_{thresh}} \leftarrow$ confidence level for occupancy
        $\mathcal{W}_r, \mathcal{W}_b, \mathcal{W}_g \leftarrow$ cached weights
Output: $\mathcal{F} \leftarrow$ 3D probabilistic field
1:  procedure UPDATESHADOWFIELD()
2:      Initialize $\mathcal{F}$ to $\mathcal{L}$
3:      for $z_i = 0$ to $z_p$ , $x_i = 0$ to $x_p$ , $y_i = 0$ to $y_p$, do
4:          $(x_{l_i}, y_{l_i}, z_{l_i}) = (l_{l_x}, l_{l_y}, l_{l_z}) + (x_i, y_i, z_i)$
5:          $(x_{g_i}, y_{g_i}, z_{g_i}) = (l_{g_x}, l_{g_y}, l_{g_z}) + (x_i, y_i, z_i)$
6:          $\mathcal{P}_{\mathcal{O}} \leftarrow \mathcal{O}(x_{g_i}, y_{g_i}, z_{g_i})$
7:          if $\mathcal{P}_{\mathcal{O}} > P_{\mathcal{O}_{thresh}}$ then
8:              $\mathcal{F}(x_{l_i}, y_{l_i}, z_{l_i}) \leftarrow (1 - P_{\mathcal{O}})$
9:          else
10:             $\mathcal{C}_r \leftarrow \mathcal{W}_r(x_i, y_i, z_i) * \mathcal{F}(x_{l_{i-1}}, y_{l_i}, z_{l_i})$
                $\mathcal{C}_b \leftarrow \mathcal{W}_b(x_i, y_i, z_i) * \mathcal{F}(x_{l_i}, y_{l_{i-1}}, z_{l_i})$
                $\mathcal{C}_g \leftarrow \mathcal{W}_g(x_i, y_i, z_i) * \mathcal{F}(x_{l_i}, y_{l_i}, z_{l_{i-1}})$
                $\mathcal{F}(x_{l_i}, y_{l_i}, z_{l_i}) \leftarrow \mathcal{C}_r + \mathcal{C}_b + \mathcal{C}_g$
11:     return $F$
```

Computational and Storage Complexities

Referring to Alg. 3, the upper bound of our run-time computational complexity is 2 additions and 3 multiplications per voxel, which scales linearly with the number of voxels and cubically with the grid resolution. This is very useful, especially when running other demanding computational processes. Since our method requires no special hardware, it may run on a separate CPU. This offloads computation from the main processor allowing it to perform demanding motion planning for a complex system with high degrees of freedom which is the case for our experiments, as shall be discussed in the hardware tests section. The storage complexity for the 3D case, excluding the occupancy grid, is that of the *shadow field* and the three weights

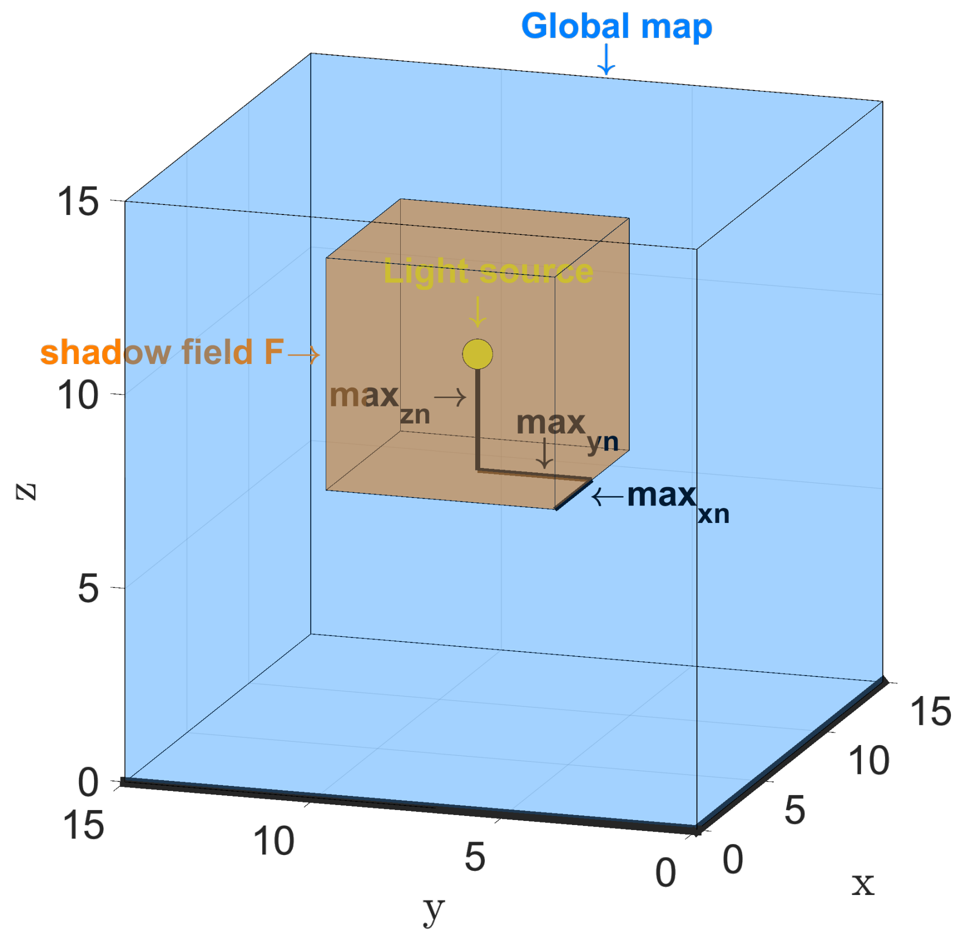


Figure 4.11: Illustration depicting the *shadow field* being a subset of the global map, the light source, and the maximum number of voxels separating the light center from the lower boundary of the *shadow field*.

distributions $\mathcal{W}_r$, $\mathcal{W}_b$, $\mathcal{W}_g$, which is exactly $4\,n_x n_y n_z$, assuming that the map has a volume of $(n_x \times n_y \times n_z)$.

We provide 3D simulations as we did in the 2D case. We pass the light dynamically in a 3D environment. The environment is shown without a light and without shadows in the first illustration in Fig. 4.12. Since the mesh for a 3D field is in 4D, we use a trick to visualize the field by project it onto an occupancy map. We specify that cells having a *shadow field* value ≤ 0.72 are occupied. In the second illustration in Fig. 4.12, the light is positioned at (19, 10, 10), and we show the cells having less than 72% visibility as occupied. This specific threshold may be varied. In the third illustration, the light position is near the center and higher than most of the occluding objects at (60, 65, 80), so the shadows are cast downwards. In the final illustration, the light is actually occluded from the viewing point, being behind one of the occluding obstacles at (70, 90, 17).

Lastly, we provide a 2D deterministic sample application of *shadow field* as well as a solution obtained using a naive-gradient descent algorithm in Appendix B.

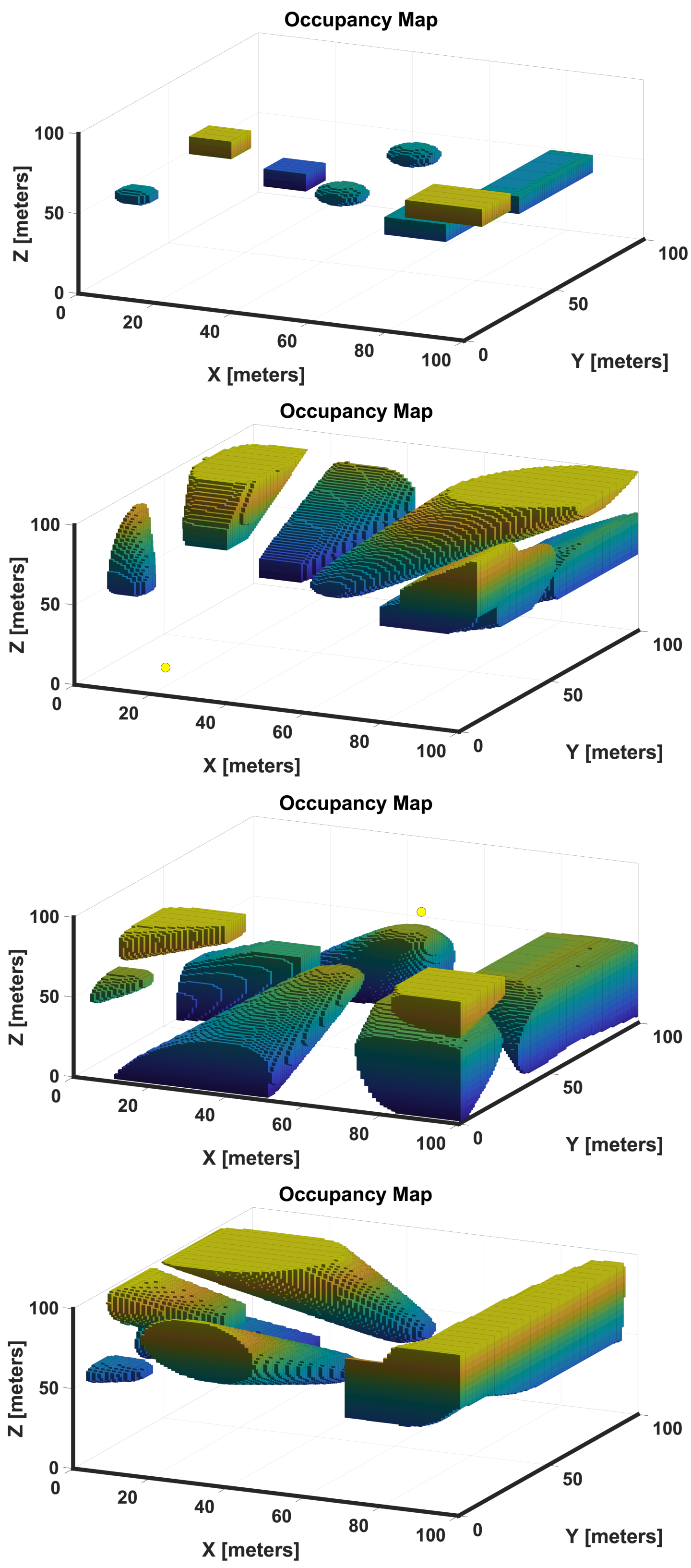


Figure 4.12: Representing shadow field cells that have a visibility ≤ 0.72 as occupied cells in an occupancy map since the *shadow field* mesh for a 3D field is in 4D.

Chapter 5

Experimental and Physics-Based Simulation Results

In this chapter, we present the experimental and physics-based simulation results that validate the formulations presented in previous chapters. All of the code required to implement the simulations and perform the experiments was done in C++ on ROS. Gazebo is the chosen physics-based simulation environment, and Pinocchio is the main tool used in kinematic and dynamic modelling.

We start off by showing the Kinova 7DOF path following simulation results for three cases; the nominal case, the case with a wrong initialization, and the case with collision avoidance. Then we present experimental results from real-time *shadow field* mapping using hardware. Lastly, we conclude with occlusion avoidance whole-body motion planning simulations for both the mobile manipulator and ALMA.

OCS2 Toolbox

The main toolbox used to implement the formulations is OCS2 [40]. It allows for efficient implementation of SLQ-MPC, among other algorithms. It provides all the necessary tools to setup the whole problem starting with system dynamics to defining costs and constraints from a URDF model due to its support of Pinocchio. It performs an automatic differentiation tool to evaluate the necessary derivatives concerning the system dynamics, constraints and costs. Lastly, it provides user-friendly tools to establish an interface with ROS.

5.1 Path Following

The first step in path following is obtaining a parametrized path. For demonstrative reasons, we use a simple numerical computation of the arclength parameter to obtain our path parametrization whereby the main goal is the formulation discussed in Section 3.1 rather than ways on how to parametrize the path. For more details about path parametrization, refer to [31]. In the C++ code written for this project, a separate module is provided that takes a parametrized path as input irrespective of the way it was parametrized. As discussed in Section 3.1, we do not parametrize the orientation, but simply constrain it to a fixed value along each path.

5.1.1 Path Generation & Numerical Arclength Parametrization

We start with the definition of the arclength

$$\theta = \int_a^b \sqrt{\frac{dx}{dt}^2 + \frac{dy}{dt}^2 + \frac{dz}{dt}^2}\, dt = \int_a^b \mid\mid v(t) \mid\mid dt, \tag{5.1}$$

where θ is our arclength parameter as detailed in 3.1, and x, y, and z are the components of the differentiable vector valued function on [a,b], $r(t) = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}$. The gradients required for (5.1) can be obtained numerically using central or forward finite differences. Integration itself may be performed using the cumulative trapezoidal method. The last step in obtaining the arclength parametrization is generating a linearly spaced vector parametrized by the resultant θ.

Vector $r(t)$ itself may be obtained by generating a sample reference trajectory using harmonic trajectories [31]. Harmonic trajectory result in a position profile that is continuously differentiable, i.e., $\in \mathcal{C}^\infty$. We provide in Fig. 5.1 example x(t), y(t), and z(t) constituting r(t). The resulting arclength parametrized path is shown in Fig. 5.2.

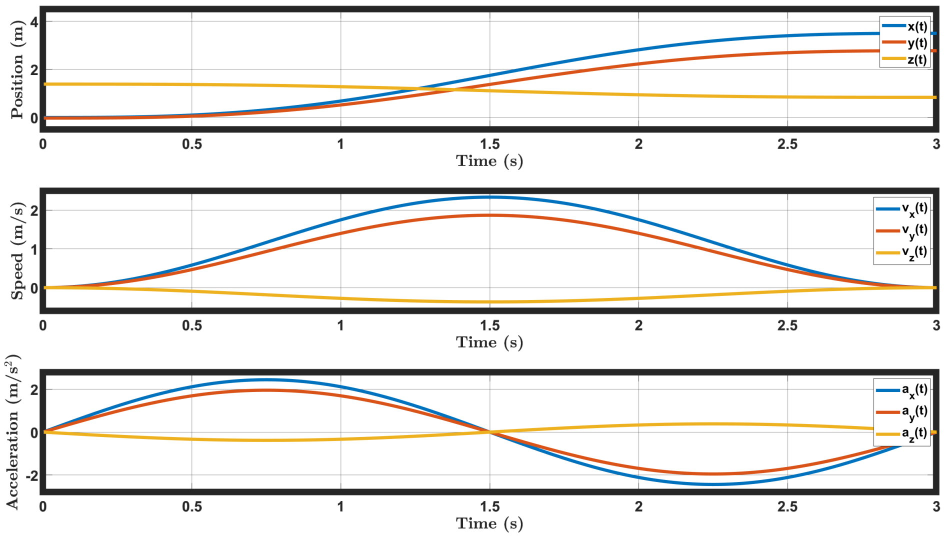


Figure 5.1: Sample reference trajectory that serves as a basis, r(t), to obtain the arclength parameterization in (5.1). Position, speed, and acceleration curves for the three dimensions are continuous and smooth due to the fact that harmonic trajectories $\in \mathcal{C}^\infty$.

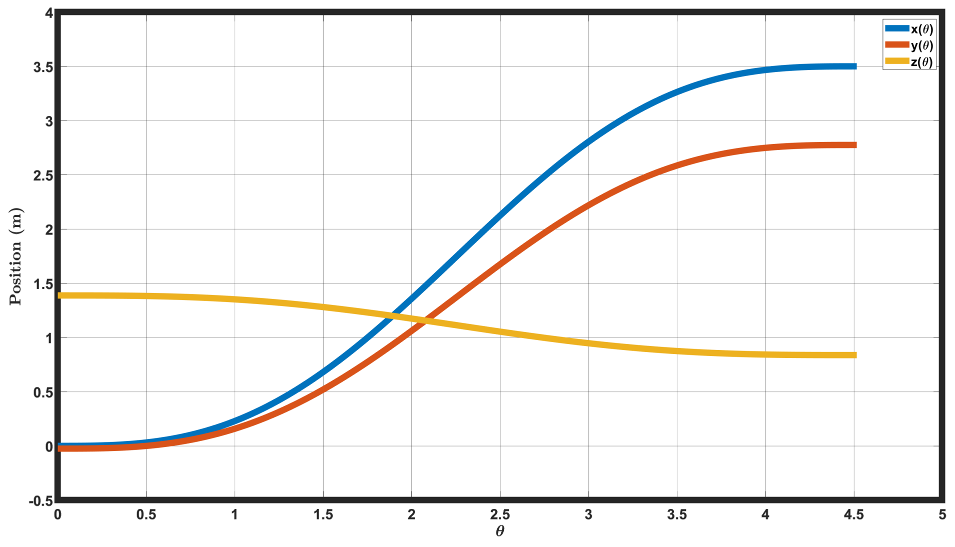


Figure 5.2: Sample reference path to be followed. The paths along the three directions, x(θ), y(θ), and z(θ) are also smooth and continuous.

To recap, our path generation module takes in as input the path resolution, which we set at 0.0025, a final time t_f, % of the trajectory where we have acceleration and % of the trajectory where we have deceleration, rise and fall respectively, the distance covered in x, y, and z, and the offset from origin/initial condition of the end effector. It then generates a trajectory, from which we extract the arclength parametrization θ. Finally, the geometric path is then arclength parametrized and passed as the desired path to be followed.

5.1.2 Nominal case

We first investigate the nominal case whereby the robot's end-effector has to follow a simple and easy path that starts from the end-effector's initial condition. In Fig. 5.3, we show the evolution of the robot end-effector from starting to ending positions along the path which is highlighted. We also show the end-effector's frame. In Fig. 5.4, we show ROS's rqt multiplots of θ in blue on the left, and ω in red on the right, both against time. Reminder that θ is the arclength parameter and ω is the rate of progression of θ with time. Nominally, those values reach their respective setpoints with no issues as the robot nominally follows the path.

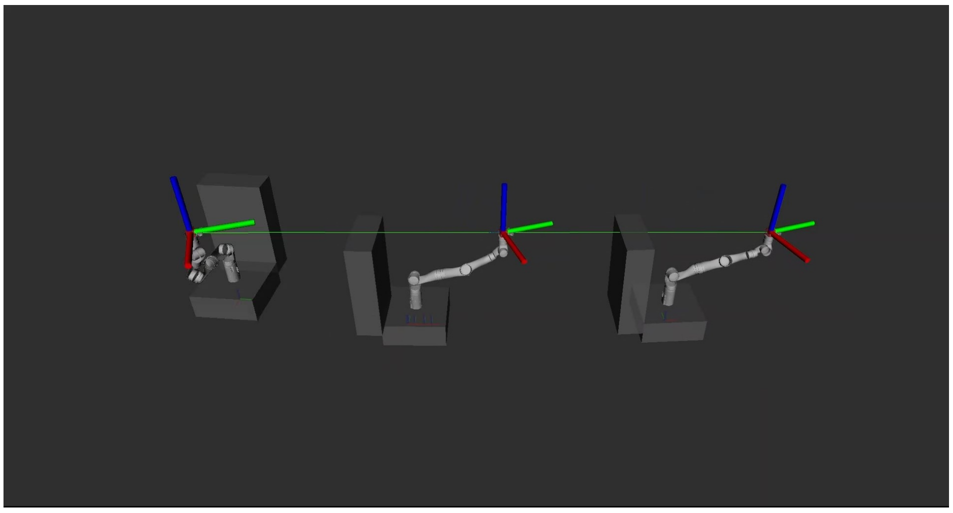


Figure 5.3: Multiplicity illustration showing the evolution of the robot end-effector along the path throughout a nominal path following setting. The target path and the end-effector frame are highlighted.

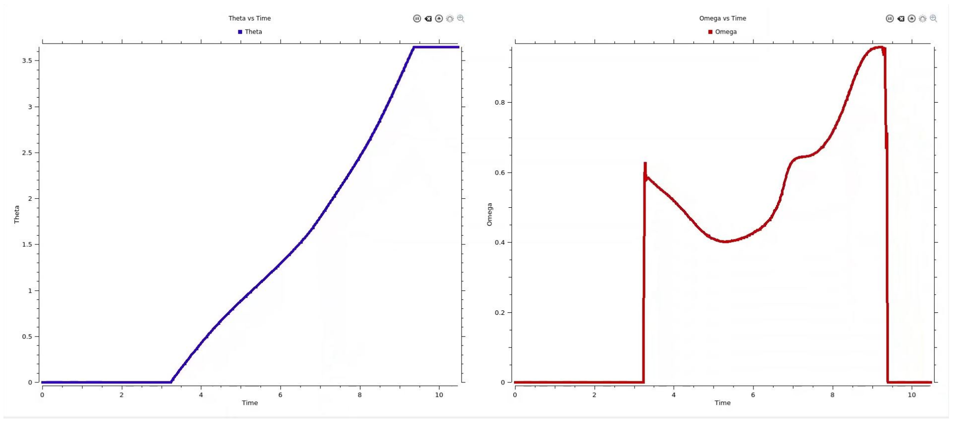


Figure 5.4: Plots of arclength θ in blue and ω in red for the nominal case.

5.1.3 Wrong Initialization Case

The second case of interest is the case where the path has an offset from the end-effector initial position, i.e., we have a wrong initialization. In Fig. 5.5, we show the robot initially moving to the path starting point, and then continuing along the path towards the ending position of the end-effector. We also show the end-effector's frame. In Fig. 5.6, the behaviour of solver can be seen where path progression ω is killed initially as progression was not favorable. ω even went negative to correct for the initial θ progression. Once the end-effector is back on track along the path, progression proceeds normally and nominally.

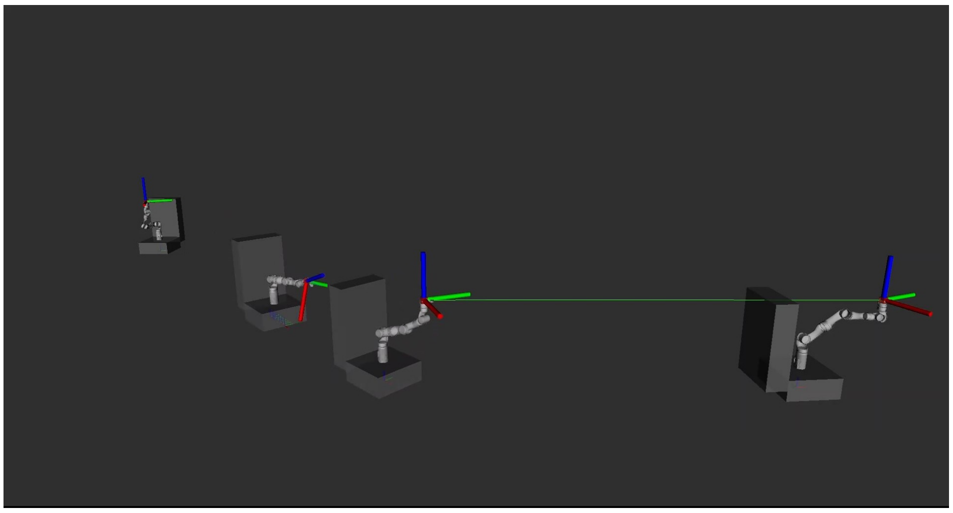


Figure 5.5: Multiplicity illustration showing the evolution of the robot end-effector along the path throughout a path following setting with a wrong initialization.

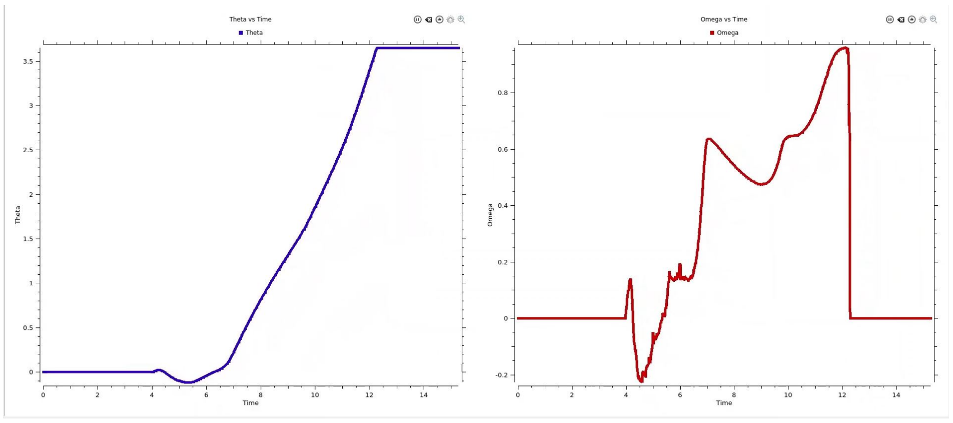


Figure 5.6: Plots of arclength θ in blue and ω in red for the case of wrong initialization. Progression freezes till end-effector is back on track.

5.1.4 Collision Avoidance Case

In the third case, we place an obstacle along the path. We can detect this obstacle's location and infer if there will be a collision with it or not along the path, and if yes, infer the value of θ at which there will be collision. We show the evolution of the behaviour of the robot in Fig. 5.7 where the robot pauses its progression right before collision, i.e., achieving collision avoidance. In Fig. 5.8, the progression is visibly killed. The solver tries to proceed at different times along the path but since progression is unfavorable, the robot's end-effector successfully avoids collision.

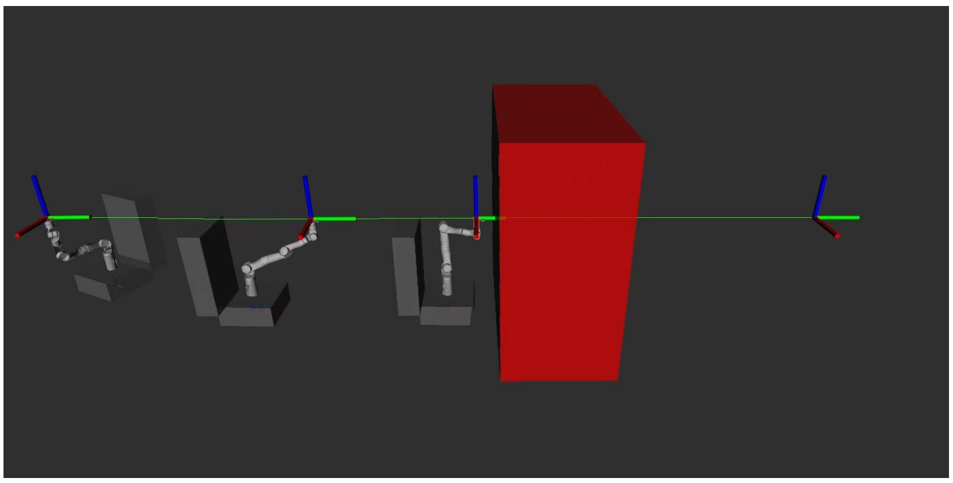


Figure 5.7: Multiplicity illustration showing the evolution of the robot end-effector along the path throughout a nominal path following setting with collision avoidance.

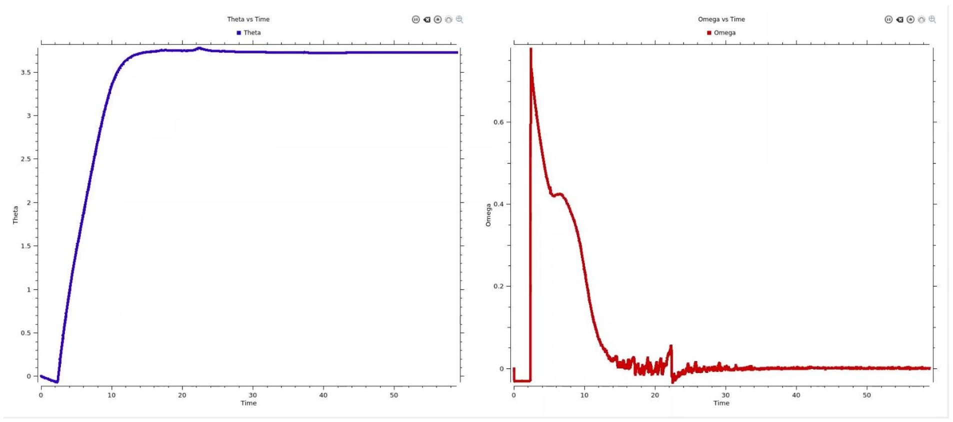


Figure 5.8: Plots of arclength θ in blue and ω in red for the case of collision avoidance. Progression freezes indefinitely right before collision.

5.1.5 Qualitative and Quantitative Discussion for the Nominal Case

In this subsection, we evaluate the qualitative and quantitative performance of our path following for the nominal case. In Fig. 5.9, we compare the three components for the desired path, $x(\theta)_{des}$, $y(\theta)_{des}$, and $z(\theta)_{des}$, with the actual path components obtained in simulation, $x(\theta)_{actual}$, $y(\theta)_{actual}$, and $z(\theta)_{actual}$ respectively. The RMSE for each of the components is 6.4mm, 6.5mm, and 5.2mm respectively. Even though our main goal is not to follow the desired path as closely as possible, we obtained sufficiently low RMSE values. The penalty assigned to following the path was high, and much lower RMSEs may be obtained if one was to tune the solver for that purpose. We add the time as a third dimension in Figs. 5.10, 5.11, and 5.12.

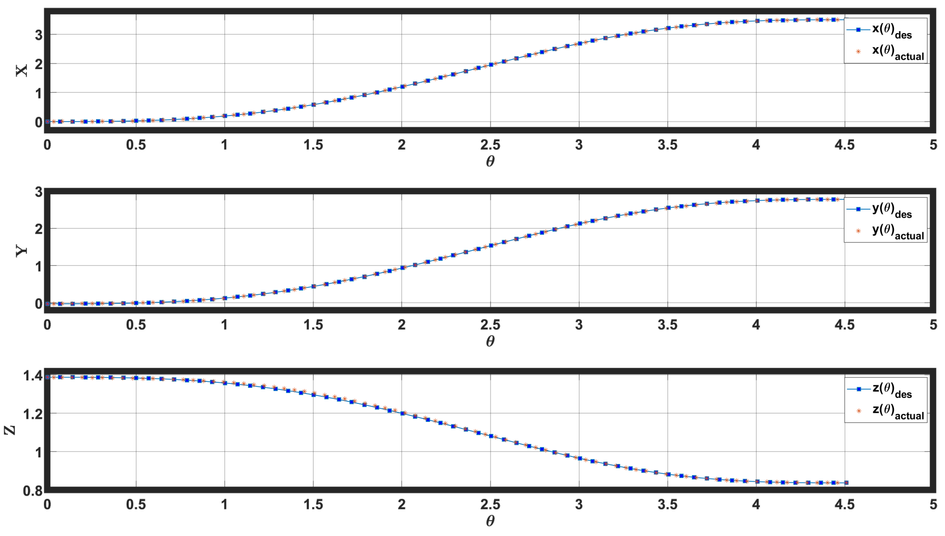


Figure 5.9: Comparison between the desired path components and the actual path components for the nominal case.

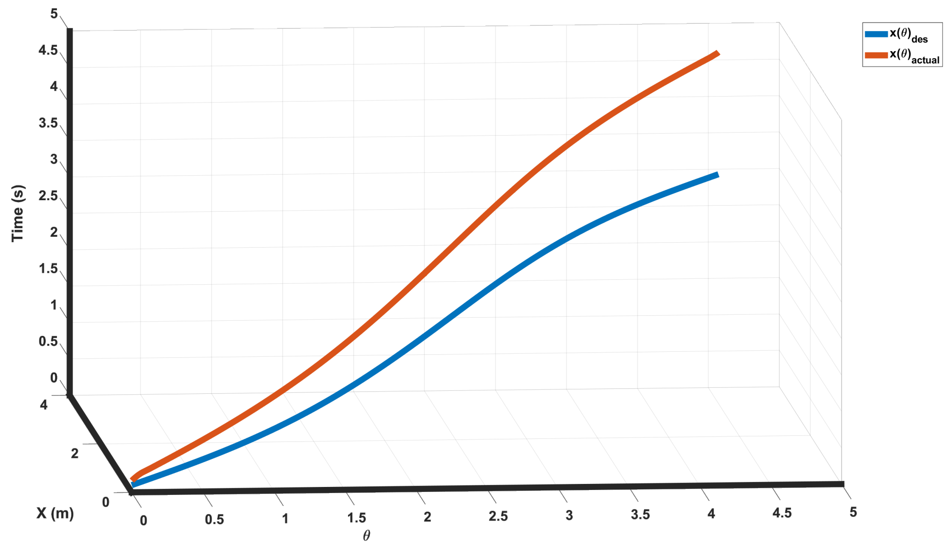


Figure 5.10: Comparison between the $x(\theta)_{des}$ and $x(\theta)_{actual}$ for the nominal case showing the time trade-off.

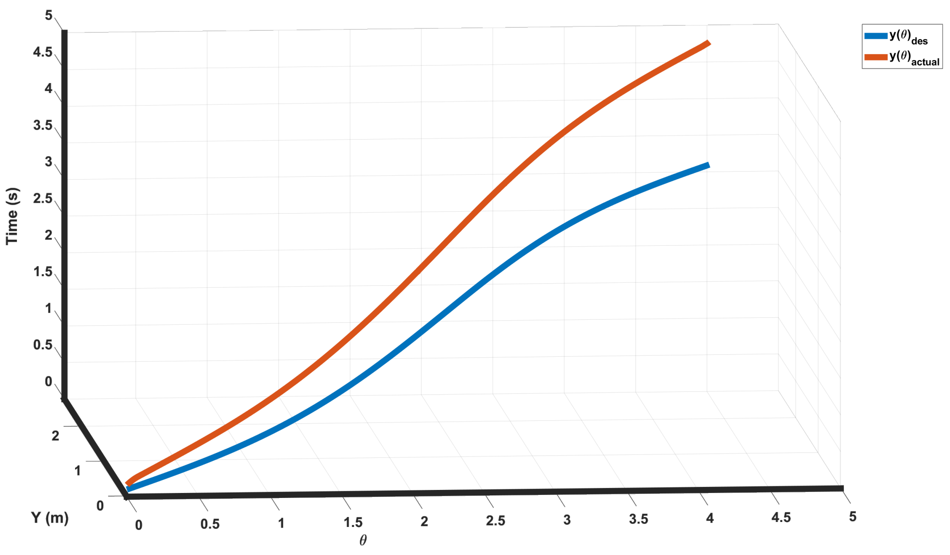


Figure 5.11: Comparison between the $y(\theta)_{des}$ and $y(\theta)_{actual}$ for the nominal case showing the time trade-off.

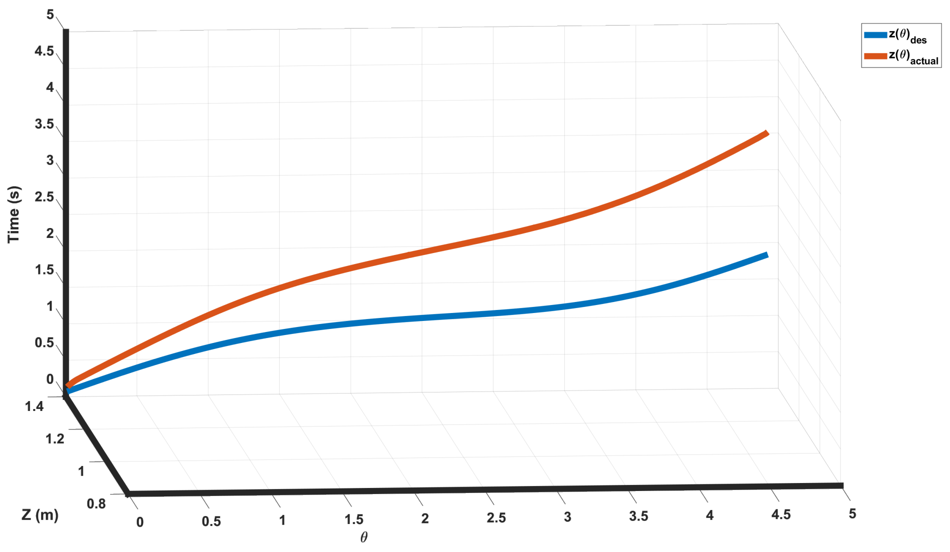


Figure 5.12: Comparison between the $z(\theta)_{des}$ and $z(\theta)_{actual}$ for the nominal case showing the time trade-off.

The RMSE with respect to time is around 0.98s, where the solver prefers to take more time for the sake of better following the path under the constraints, therefore exploiting the DOF in time. This degree of freedom is also tunable in our formulation as we can increase the penalty on following a desired path progression $\omega = 1$.

5.2 Dynamic Occlusion Avoidance Within Whole-Body Motion Planning

5.2.1 Motion Planning and Control Pipeline

In **Chapter 2**, Section 2.4, we introduced ALMA's equations of motion and background on SLQ-MPC and in **Chatper 3**, Section 3.2 we formulated the cost function at hand and presented the necessary equality and inequality constraints. Those are fundamental for whole-body motion planning, but in terms of execution and control, we need to briefly introduce the whole-body controller (WBC) that performs the actual tracking of the planned motion while prioritizing some tasks as presented in Fig. 5.13.

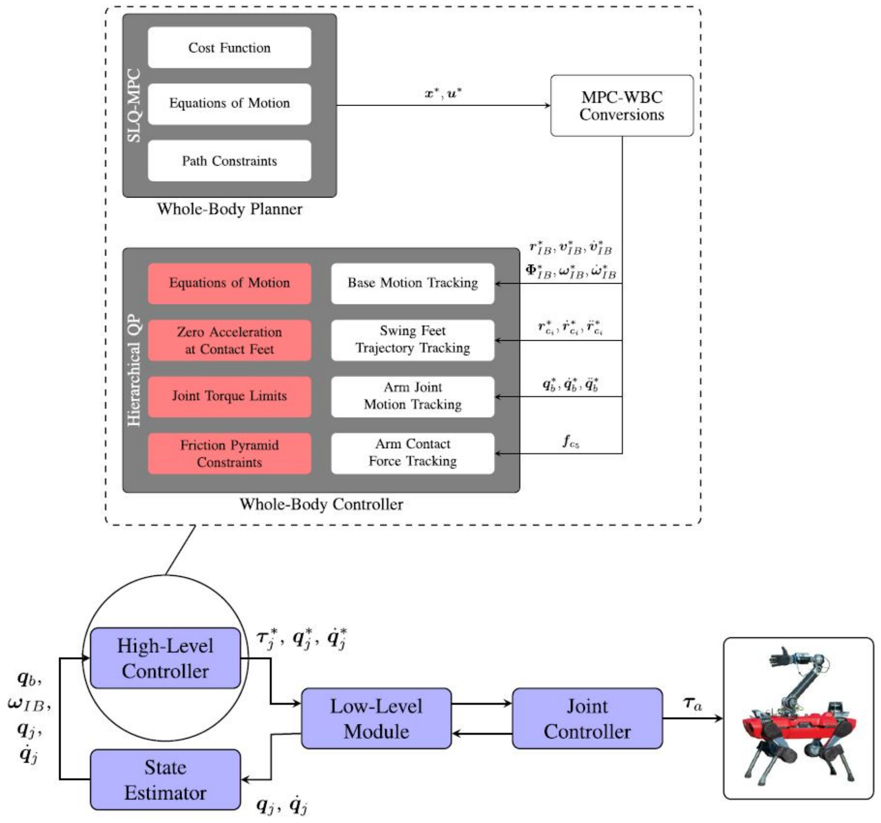

SLQ-MPC
Cost Function
Equations of Motion
Path Constraints
Whole-Body Planner
x^*, u^*
MPC-WBC Conversions
Hierarchical QP
Equations of Motion
Zero Acceleration at Contact Feet
Joint Torque Limits
Friction Pyramid Constraints
Base Motion Tracking
Swing Feet Trajectory Tracking
Arm Joint Motion Tracking
Arm Contact Force Tracking
$r^*_{IB}, v^*_{IB}, \dot{v}^*_{IB}$
$\Phi^*_{IB}, \omega^*_{IB}, \dot{\omega}^*_{IB}$
$r^*_{c_i}, \dot{r}^*_{c_i}, \ddot{r}^*_{c_i}$
$q^*_b, \dot{q}^*_b, \ddot{q}^*_b$
f_{c_5}
Whole-Body Controller
High-Level Controller
$\tau^*_j, q^*_j, \dot{q}^*_j$
Low-Level Module
Joint Controller
τ_a
$q_b, \omega_{IB}, q_j, \dot{q}_j$
State Estimator
$q_j, \dot{q}_j$

Figure 5.13: The high-level controller in ALMA's control architecture consists of a whole-body planner that interacts with a whole-body controller that prioritizes the tasks indicated in red [35].

Those tasks are solved as a hierarchical QP problem that optimizes for generalized accelerations and contact forces. The torques may then be obtained from those accelerations and forces using inverse dynamics. The WBC tracks reference motion induced on the base by said forces rather than tracking the forces themselves. Moreover, it relies on a more realistic model than the planner, so it imposes stricter conditions on physical correctness. It is also noteworthy that MPC planner solutions for the arm contact forces are treated differently than the rest of the forces since the dynamic model adopted by WBC does not include the object dynamics (2.18).

Lastly, Fig. 5.13 includes a MPC-WBC conversions step, where MPC solutions are coupled with the WBC tracking tasks. Actuated joint accelerations $\ddot{q}_j$ are computed using finite differences while the DynaArm's joint positions and velocities as well as contact forces are obtained directly from the MPC state and input solutions. Trajectories for feet swings are obtained using basic kinematic transformations and using the following expression for accelerations

$$\ddot{r}_{c_i} = J_{c_i}\ddot{q}_j + \dot{J}_{c_i}\dot{q}_j. \tag{5.2}$$

The unactuated base pose is part of the MPC state solution, whereas the base's linear and angular velocities are extracted from (2.15) with a mapping equivalent to (A.10),

$$\begin{bmatrix} v_{IB} \\ \omega_{IB} \end{bmatrix} = \begin{bmatrix} \mathbb{I} & 0_{3\times3} \\ 0_{3\times3} & T(\Phi^{zyx}_{IB}) \end{bmatrix} \dot{q}_b, \tag{5.3}$$

where $T(\Phi^{zyx}_{IB})$ play a similar role as (A.12) but for a ZYX-Euler representation rather than a roll, pitch, and yaw one.

The base feedforward accelerations are obtained by time-differentiating (2.15), obtaining

$$\ddot{q}_b = A_b^{-1}\Big(\dot{h}_{com} - \dot{A}\dot{q} - A_j\ddot{q}_j\Big), \tag{5.4}$$

where $\dot{h}_{com}$ is obtained in (2.13), and $\dot{A}\dot{q}$ is computed by zeroing the joint accelerations within a recursive Newton-Euler algorithm and transforming the resulting twist from a base one to centroidal one. The accelerations $\dot{v}_{IB}$ and $\dot{\omega}_{IB}$ are computed as

$$\begin{bmatrix} \dot{v}_{IB} \\ \dot{\omega}_{IB} \end{bmatrix} = \begin{bmatrix} \mathbb{I} & 0_{3\times3} \\ 0_{3\times3} & T \end{bmatrix} \ddot{q}_b + \begin{bmatrix} \mathbb{I} & 0_{3\times3} \\ 0_{3\times3} & \dot{T} \end{bmatrix} \dot{q}_b. \tag{5.5}$$

The MPC runs on one of the two onboard computers having an Intel Core i7-8850H CPU @4 GHz hexacore processor at a rate of 70Hz in free motion for a time horizon of 1 second. The planner and the WBC receive feedback from encoder and IMU sensor fusion estimator to estimate the base pose. The feedback loop between the state estimator and the WBC runs at 400Hz, whereas the low-level module interacts with the joint controller at 2.5kHz, whereby the torque commands are generated as

$$\tau_a = \tau_j^* + K_p(q_j^* - q_j) + K_d(\dot{q}_j^* - \dot{q}_j). \tag{5.6}$$

5.2.2 Realtime *Shadow Field* Mapping

To validate our *shadow field*, two real-time mapping experiments are considered. In each of them, ALMA is placed in a different cluttered environment, and data from the Veldoyne sensor are used to construct a 3D probabilistic occupancy grid. The occupancy grid is then used to compute and publish a 3D *shadow field* spanning $16\times16\times2m^3$ at a resolution of 1000 voxels per cubed meter. The onboard mapping computer runs the whole pipeline at rates exceeding 100Hz, far beyond the Velodyne pointcloud update rate ($\sim$15Hz). Since it is not possible to visualize the whole 3D *shadow field* in a meaningful manner, we visualize only a horizontal slice of it at a defined height in the form of a pointcloud having gray-scale intensities proportional to the values of the *shadow field*. At that slice level, brighter areas represent visible regions while darker areas represent occluded regions. We publish this slice at the robot's end-effector level. In both experiments, we coincide the light position with the end-effector position at the center of the field, i.e., we are evaluating the end-effector's sight of the surrounding scene. We also visualize the occupied voxels, which have different colors correlated to their occupation probability. We show the top view of the slice, the occupied voxels, and the Velodyne pointcloud in Fig. 5.14. Asides from noise arising at the stage of building the occupancy grid, the resulting shadows retain the same smoothness and continuity characteristics as the soft shadow presented in Fig. 4.6.

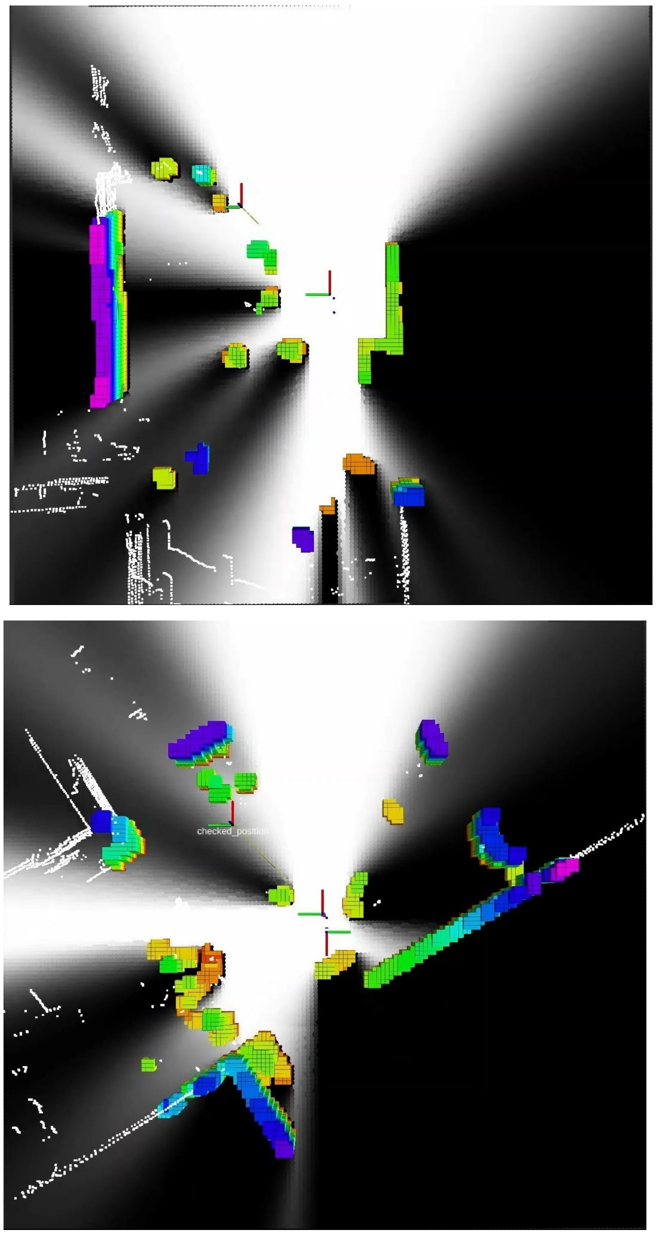


Figure 5.14: Top view of the occupied voxels (colored), the Velodyne pointcloud (small white dots), and the resulting *shadow field* pointcloud slice (gray-scale). The light source and the end-effector are at the center of each image. The umbras produced by the occluding voxels are dark while their penumbras are the shades of gray.

5.2.3 Physics-based Simulations

The final contribution of this paper is experiments running in a physics-based simulation environment, Gazebo, that validate our ALMA motion planning and control pipeline. The MPC is solved at a 1.0s horizon, and the relevant penalties arise from the actuator inputs, the visibility and orientation costs, height tracking, and the actuator speed, position, and torque limits. Other regularization costs and gait-related constraints are active during this process. We also provide the simulation with a given occupancy grid built based on simulated obstacles in the environment. The grid resolution we adopt is 1000 voxels per cubed meter.

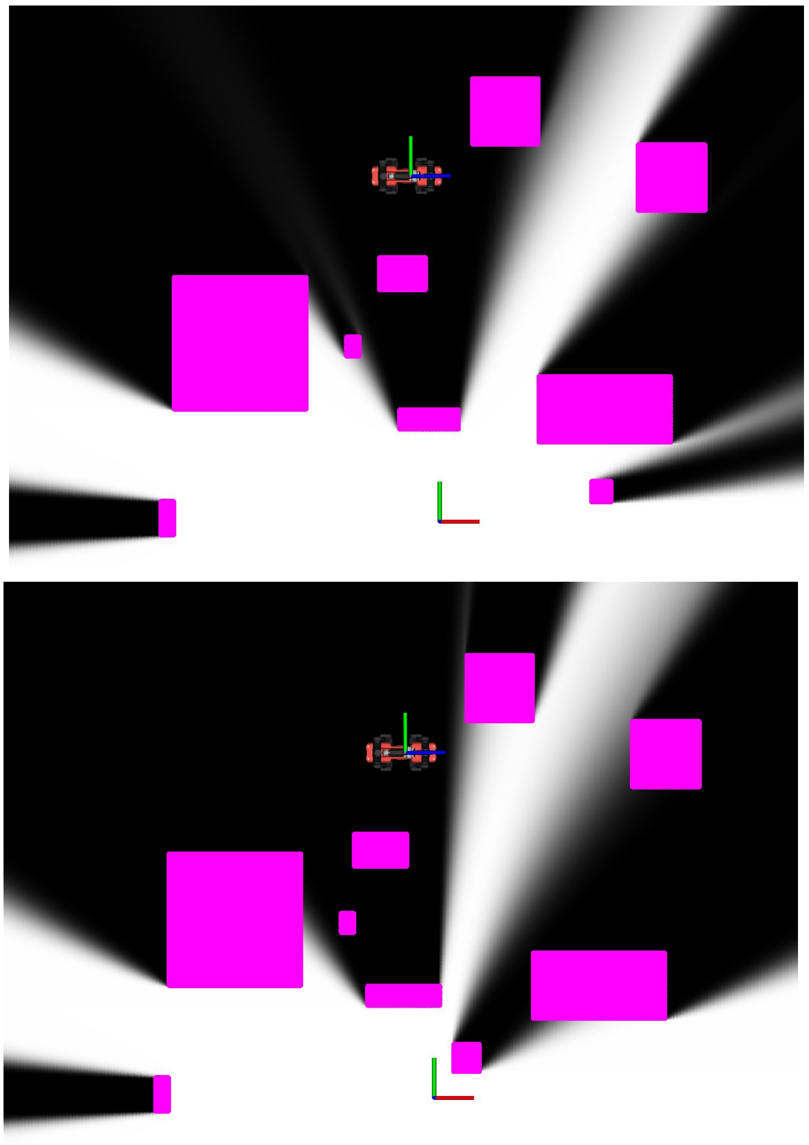


Figure 5.15: Top view of two sample scenes where we run our simulations. The robot end-effector, ALMA's in this case, has to establish line-of-sight with the target frame, hidden behind obstacles (violet). Shadows cast here have been generated using our proposed algorithms, so we note umbras and penumbras here too.

In Fig. 5.15, we illustrate the top view of each of the two scenes in which we carry out the simulation. In both illustrations, a slice of the *shadow field* at the end-effector level is shown in gray-scale with the same intensity scale as previously introduced for Fig. 5.14. Umbras and penumbras described there can also be noted here. As shown in Fig. 5.15, we make sure that the end-effector starts in an occluded position. The robot must then plan a least-shadowy path for the end-effector position that leads it to establish line-of-sight with the light position.

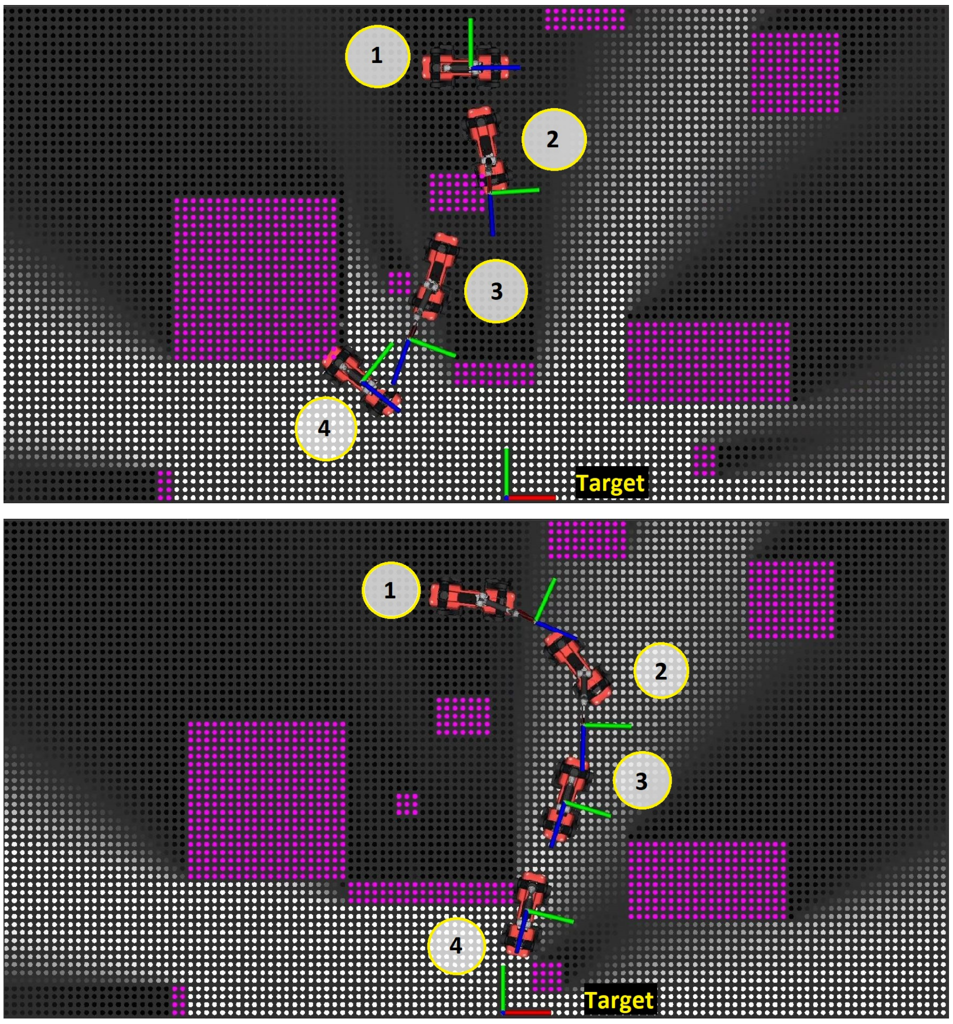


Figure 5.16: Multiplicity illustrations of our physics-based simulations in the two scenes introduced in Fig. 5.15. The solver solution in each simulation is reflected through 4 different snapshots of the robot. The target is tagged. Violet points correspond to obstacles at the level of the end-effector..

In Fig. 5.16, we illustrate the robot's motion and its end-effector frame pose in a top view. In the first simulation, snapshot 1 shows the robot in its starting occluded position. The second snapshot shows the end-effector sliding along the surface of an occluding obstacle so as to circumnavigate it. The third snapshot shows the end-effector being extended ahead of the robot, trying to reach locations with higher visibility. The last snapshot shows the robot in a steady state. Having reached full visibility, the end-effector is free to lock onto the target. We achieve an orientation error complement of 0.995. In the second illustration, we also begin in an occluded position. The differences in this scene are minor relative to the first one, where we block the path of the previous solution by extending the first obstacle facing the light frame. We also move one of the obstacles and place it right next to the light position. The robot, as a result, directly extends the end-effector into the visible region in snapshot 1, then traverses the slightly shadowed region and peaks its end-effector through the tight passage by the fourth snapshot to achieve full visibility. As an interesting emergent behavior, the end-effector avoids collisions with obstacles since their *shadow field* values are zero. This behavior may be extended to other robot frames.

In a game of hide and seek, our robot manages to establish 100% visibility with the target frame while locking completely to it after we hide our target of interest behind several simulated occluding obstacles. Note that those scenarios are extreme. We will generally not face a problem where the robot starts in complete invisibility or where the line-of-sight between the target and the end-effector is suddenly hindered by several obstacles that completely block visibility. More realistic scenarios include partial occlusion, where the robot end-effector has to adjust a bit so as to maintain sight, as is the case with humans and animals when their line-of-sight to a target is blocked. After all, this was the goal all along, to mimic human intelligent behaviour in such scenarios.

It is also noteworthy that the proposed dynamic occlusion avoidance scheme is a gradient-based technique, and such planning techniques cannot return a viable solution in case the local gradient that is reachable along the horizon is flat. In such a case, a higher level planner is required where the robot enters a frontier-based exploration phase or backtracks to the last position where there was visibility or even change the task completely. Moreover, the behaviour of the robot end-effector in this scheme directly depends on the quality of the probabilistic occupancy map, which itself is bottlenecked by the LIDAR's or the RGB-D's update rate. Lastly, the behaviour of the robot also depends on the mapping resolution, which itself increases the complexity of the scheme the lower it gets, otherwise trilinear interpolation might not return high fidelity *shadow field* values and gradients, causing as a result a jittery end-effector movement.

We provide more simulations results for both Kinova 7DOF mobile manipulator and ALMA in Appendix B.

Chapter 6

Future Work

As mentioned at the end of Section 5.2.3, our proposed dynamic occlusion avoidance formulation will not work for extreme scenarios where the end-effector is initialized in complete darkness deep within an occluding obstacle's umbra, nor will it work in the extreme case where sudden complete occlusion takes place. A higher level control is required to take over in that case. Moreover, what is still missing is hardware experiment having the same setup as the physics-based simulations to validate occlusion avoidance in real life. Lastly, further tests and tuning is required to obtain robustness in real life. The occupancy map provided for the simulation was noise-free, making it unrealistic.

Regarding our path following formulation, we are missing hardware tests the Kinova 7DOF mobile manipulator. One may formulate the same SLQ-MPC path following problem for ALMA then test it in simulation and on hardware as a second step. The formulation itself is also missing a minimal parametrization of the orientation. One may add speed related constraints so as to ensure stability of the end-effector, and one may also experiment with a speed-assigned as [6] did. We reiterate that in this context, a speed assigned path following problem differs from a trajectory tracking one where the prior is similar to scaling the timing law on the go within motion planning.

Chapter 7

Conclusion

In this thesis, we formulated a constrained path following SLQ MPC problem for a mobile manipulator. We embedded the arclength variable with which the desired path is parametrized within the system states, giving the solver full control over the rate of progression as it deems fit to accommodate for arising costs and path constraints. We implemented this formulation for a 7DOF manipulator and we studied its behaviour for three cases. For the nominal case, we provided a qualitative and quantitative description of the manipulator's path following performance. For the case of a wrong initialization, we highlighted the solver's ability to correct the initial error by backtracking the solution of the arclength and proceeding nominally once the end-effector is back on track. For the last case, we show the solver's ability to avoid collisions by placing a relaxed-log barrier at the arclength value where a collision is expected to happen.

In this work, we also proposed an MPC formulation based on visibility constraints. We augmented our motion planning cost function with a penalty maximizing relaxed log-likelihood of visibility probability. We introduced a probabilistic *shadow field* that quantifies visibility probability based on the occupancy map of the scene. We validated the quality of this map in simulation and hardware. We further discussed the computational and storage complexities of our *shadow field* mapping and showcased its computational efficiency for onboard applications by real-time mapping on ALMA hardware. A comparison between hard and soft shadows for one 2D implementation shows the extent and accuracy of our approximation. Last but not least, we demonstrated the validity of our proposed MPC formulation for motion planning and dynamic occlusion avoidance in simulation for ALMA.

Bibliography

[1] M. H. Arbo, E. I. Grotli, and J. T. Gravdahl, "Mid-level mpc and 6 dof output path following for robotic manipulators," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 2017, pp. 450–456.

[2] ——, "On model predictive path following and trajectory tracking for industrial robots," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, 2017, pp. 100–105.

[3] N. van Duijkeren, R. Verschueren, G. Pipeleers, M. Diehl, and J. Swevers, "Path-following nmpc for serial-link robot manipulators using a path-parametric system reformulation," in *2016 European Control Conference (ECC)*, 2016, pp. 477–482.

[4] T. K. Jespersen, M. al Ahdab, J. d. Dios F. Mendez, M. R. Damgaard, K. D. Hansen, R. Pedersen, and T. Bak, "Path-following model predictive control of ballbots," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1498–1504.

[5] S. Yu, Y. Guo, L. Meng, T. Qu, and H. Chen, "Mpc for path following problems of wheeled mobile robots," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 247–252, 2018, 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318326752

[6] T. Faulwasser, T. Weber, P. Zometa, and R. Findeisen, "Implementation of nonlinear model predictive path-following control for an industrial robot," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1505–1511, 2017.

[7] G. Allibert, E. Courtial, and F. Chaumette, "Predictive control for constrained image-based visual servoing," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 933–939, 2010.

[8] M. Sauvee, P. Poignet, E. Dombre, and E. Courtial, "Image based visual servoing through nonlinear model predictive control," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 1776–1781.

[9] G. Allibert, E. Courtial, and F. Chaumette, "Visual servoing via Nonlinear Predictive control," in *Visual Servoing via Advanced Numerical Methods*, Chesi, G., Hashimoto, and K., Eds. LNCIS 401, Springer-Verlag, 2010, pp. 375–394. [Online]. Available: https://hal.inria.fr/inria-00548935

[10] Y. Mezouar and F. Chaumette, "Path planning for robust image-based control," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 534–549, 2002.

[11] M. Kazemi, K. Gupta, and M. Mehrandezh, *Path-Planning for Visual Servoing: A Review and Issues.* London: Springer London, 2010, pp. 189–207. [Online]. Available: https://doi.org/10.1007/978-1-84996-089-2_11

[12] S. Zhang, H. He, Y. Zhang, X. Li, and Y. Sang, "Dynamic self-occlusion avoidance approach based on the depth image sequence of moving visual object," *Mathematical Problems in Engineering*, vol. 2016, pp. 1–11, 01 2016.

[13] V. Lippiello, B. Siciliano, and L. Villani, "An occlusion prediction algorithm for visual servoing tasks in a multi-arm robotic cell," in *2005 International Symposium on Computational Intelligence in Robotics and Automation*, 2005, pp. 733–738.

[14] I. Karakostas, V. Mygdalis, A. Tefas, and I. Pitas, "Occlusion detection and drift-avoidance framework for 2d visual object tracking," *Signal Processing: Image Communication*, vol. 90, p. 116011, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0923596520301673

[15] T. Nägeli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges, "Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1696–1703, 2017.

[16] C. McGreavy, L. Kunze, and N. Hawes, "Next best view planning for object recognition in mobile robotics," in *PlanSIG*, 2016.

[17] K. Wu, R. Ranasinghe, and G. Dissanayake, "Active recognition and pose estimation of household objects in clutter," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4230–4237.

[18] R. Zeng, Y. Wen, W. Zhao, and Y.-J. Liu, "View planning in robot active vision: A survey of systems, algorithms, and applications," *Computational Visual Media*, vol. 6, no. 3, pp. 225–245, Sep 2020. [Online]. Available: https://doi.org/10.1007/s41095-020-0179-3

[19] J. Santos, M. Oliveira, R. Arrais, and G. Veiga, "Autonomous scene exploration for robotics: A conditional random view-sampling and evaluation using a voxel-sorting mechanism for efficient ray casting," *Sensors*, vol. 20, no. 15, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/15/4331

[20] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," *Proceedings of EuroGraphics*, vol. 87, 08 1987.

[21] F. Kraus and K. Dietmayer, "Uncertainty estimation in one-stage object detection," *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Oct 2019. [Online]. Available: http://dx.doi.org/10.1109/ITSC.2019.8917494

[22] D. Hall, F. Dayoub, J. Skinner, H. Zhang, D. Miller, P. Corke, G. Carneiro, A. Angelova, and N. Sünderhauf, "Probabilistic object detection: Definition and evaluation," 2020.

[23] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," 2020.

[24] A. Tandon and K. Karlapalem, "Medusa: Towards simulating a multi-agent hide-and-seek game," in *IJCAI*, 2018.

[25] F. Bungiu, M. Hemmer, J. Hershberger, K. Huang, and A. Kröller, "Efficient computation of visibility polygons," 2014.

[26] L. Barba, M. Korman, S. Langerman, and R. I. Silveira, "Computing a visibility polygon using few variables," 2013.

[27] V. Chvátal, "A combinatorial theorem in plane geometry," *Journal of Combinatorial Theory, Series B*, vol. 18, no. 1, pp. 39–41, 1975. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0095895675900611

[28] M. E. Newell, R. G. Newell, and T. L. Sancha, "A solution to the hidden surface problem," in *Proceedings of the ACM Annual Conference - Volume 1*, ser. ACM '72. New York, NY, USA: Association for Computing Machinery, 1972, p. 443–450. [Online]. Available: https://doi.org/10.1145/800193.569954

[29] L. Sciavicco and B. Siciliano, "Modelling and control of robot manipulators," *Measurement Science and Technology*, vol. 11, p. 1828, 11 2000.

[30] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[31] P. Rocco, "Control of industrial robots," Lecture Notes, 2021, presented to 090914, Politecnico di Milano, Milan, Italy.

[32] M. Hutter and R. Siegwart, "Robot dynamics," Lecture Notes, 2018, presented to 151-0851-00L Robot Dynamics, ETH Zurich, Zurich, Switzerland.

[33] L. Fagiano, "Constrained numerical optimization for estimation and control," Lecture Notes, 2020, presented to 052358, Politecnico di Milano, Milan, Italy.

[34] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *SII 2019 - International Symposium on System Integrations*, Paris, France, Jan. 2019. [Online]. Available: https://hal.laas.fr/hal-01866228

[35] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, "A unified mpc framework for whole-body dynamic locomotion and manipulation," 2021.

[36] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1398–1404.

[37] A. Sideris and J. Bobrow, "An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems," in *Proceedings of the 2005, American Control Conference, 2005.*, 2005, pp. 2275–2280 vol. 4.

[38] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *ICRA*. IEEE, 2017, pp. 93–100.

[39] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback mpc for torque-controlled legged robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4730–4737.

[40] "OCS2: An open source library for optimal control of switched systems," [Online]. Available: https://github.com/leggedrobotics/ocs2.

[41] *The OpenCV Reference Manual*, 4th ed., Itseez, July 2021.

Appendix A

Text Appendix

A.1 Fixed Base Kinematics

Figure A.1: 6DOF industrial fixed base manipulator, the ABB IRB 6400. Source: ABB.

For a fixed base system, like an industrial manipulator depicted in Fig. A.1, setting our robot generalized coordinate vector as the chosen degrees of freedom provided by our joints, we would obtain the *minimal set of generalized coordinates* that fully describe the configuration of our system,

$$\mathbf{q} = \begin{pmatrix} q_1 \\ \vdots \\ q_{n_j} \end{pmatrix}. \tag{A.1}$$

For every value of this set, our robot end-effector assumes a certain pose in task-space,

$$\mathbf{x}_e = \begin{pmatrix} r_e \\ \phi_e \end{pmatrix} \in \mathbb{R}^3 \times SO(3), \tag{A.2}$$

where r_e is the end-effector position and ϕ_e is the end-effector orientation, the latter being an abstraction that may be represented using different schemes (ex: Euler angles, angle-axis, unit quaternions).

The mapping from the generalized coordinates $\mathbf{q}$ to the end-effector pose $\mathbf{x}_e$

$$\mathbf{x}_e = \mathbf{x}_e(\mathbf{q}), \tag{A.3}$$

is described by forward kinematics and can be obtained for the chain of bodies constituting our robot as:

$$T_e^I(\mathbf{q}) = T_0^I \cdot \prod_{k=1}^{n_j} T_k^{k-1}(q_k) \cdot T_e^{n_j}, \tag{A.4}$$

where $T_e^I(\mathbf{q})$ represents the homogeneous transformation matrix between inertial frame and the end-effector frame, T_0^I and $T_e^{n_j}$ are static transformation matrices from inertial frame to the robot base frame and from last robot frame to the end-effector frame respectively.

Using the relationship above, one may obtain the position and rotation matrix at any frame of interest with respect to the inertial frame as

$$T_k^I = \begin{bmatrix} R_k^I & r_k^I \\ 0^T & 1 \end{bmatrix}, \tag{A.5}$$

where T_k^I is a 4×4 matrix, R_k^I is a rotation matrix that would orient frame k to the inertial frame I, r_k^I is the vector between frames k and I, and 0^T is a $1x3$ vector.

In order to obtain velocity of the end effector pose

$$\mathbf{x}_e = \begin{pmatrix} r_e \\ \phi_{m_e} \end{pmatrix}, \tag{A.6}$$

where ϕ_{m_e} is the minimal representation of the orientation, we use differential forward kinematics,

$$\dot{\mathbf{x}}_e = J_A(\mathbf{q})\dot{\mathbf{q}}, \tag{A.7}$$

where J_A represents the $m \times n_j$ analytical end-effector Jacobian where m depends on our parameterization, being 6 in case of a parametrization $\in \mathbb{R}^3 \times SO(3)$,

$$J_A(\mathbf{q}) = \begin{bmatrix} \frac{\partial \mathbf{x}_1}{\partial q_1} & \cdots & \frac{\partial \mathbf{x}_1}{\partial q_{n_j}} \\ \vdots & \ddots & \cdots \\ \frac{\partial \mathbf{x}_m}{\partial q_1} & \cdots & \frac{\partial \mathbf{x}_m}{\partial q_{n_j}} \end{bmatrix}. \tag{A.8}$$

The analytical Jacobian relates joint space and operational space by reflecting the contribution of each joint on end-effector pose. Clearly, the Jacobian depends on the parametrization. Moreover, the analytical Jacobian maps to velocities having a minimal representation of the orientation, and is actually composed of a positional Jacobian and an angular one

$$J_A = \begin{bmatrix} J_{A_P} \\ J_{A_R} \end{bmatrix}, \tag{A.9}$$

The geometric Jacobian J which maps a pose with non-minimal orientation representation is related to the analytical Jacobian in the case of a roll, pitch, and yaw representation of orientation and a $m \times n_j \equiv 6 \times 6$ as follows,

$$J_A = \begin{bmatrix} 1_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & T_A^{-1} \end{bmatrix} J, \tag{A.10}$$

where T_A is a matrix mapping between minimal and non-minimal orientation representations of the angular velocity as

$$\dot{\phi}_e = T_A(r,p,y)\dot{\phi}_{m_e} = T_A(r,p,y) \begin{pmatrix} \dot{r} \\ \dot{p} \\ \dot{y} \end{pmatrix}, \tag{A.11}$$

$\dot{r}$, $\dot{p}$, and $\dot{y}$ being the time-derivatives of the roll, pitch, and yaw respectively, and T_A being

$$T_A(r,p,y) = \begin{pmatrix} 1 & 0 & sin(p) \\ 0 & cos(r) & -cos(p)sin(r) \\ 0 & sin(r) & cos(p)cos(r) \end{pmatrix}. \tag{A.12}$$

A.2 Fixed Base Dynamics

The Lagrangian $\mathcal{L}$ of a mechanical system is defined as the difference of between its kinetic energy $\mathcal{T}$ and its potential energy $\mathcal{U}$

$$\mathcal{L} = \mathcal{T} - \mathcal{U}. \tag{A.13}$$

Using our defined set of generalized coordinates $\mathbf{q}$, we may write the Euler-Lagrange equation of the second kind as

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}_i} - \frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = f_i, \qquad \forall i = \{1, ..., n_q\}, \tag{A.14}$$

where f_i represents the forces along the components of the generalized coordinates which are not explained by conservative effects.

Kinetic Energy Computation

In order to compute the kinetic energy of the system, it is enough to add the contributions of each of the rigid bodies of the system. By applying Konig's Theorem, the contribution of each rigid body can be obtained by adding its translational and rotational energies. Kinetic energy, in basic mechanics, for each body can be expressed as

$$\mathcal{T}_i = \frac{1}{2} m_i \dot{\boldsymbol{p}}_{\boldsymbol{i}}{}^T \dot{\boldsymbol{p}}_{\boldsymbol{i}} + \frac{1}{2} \boldsymbol{\omega}_{\boldsymbol{i}}{}^T R_i I_i^i R_i^T \boldsymbol{\omega}_{\boldsymbol{i}}, \qquad \forall i = \{1, ..., n_q\}, \tag{A.15}$$

where $\dot{\boldsymbol{p}}_{\boldsymbol{i}}$ is the velocity of the center of mass of each body with respect to a global reference frame, m_i is the mass of each body, $\boldsymbol{\omega}_{\boldsymbol{i}}$ is the angular velocity of each body with respect to the same global reference frame, R_i are appropriate rotation matrices aligning the global to the local frames and I_i^i is the inertia tensor of the rigid body.

We need to express the previous equations in terms of the coordinates $\mathbf{q}$, so we apply the chain rule on the vectors $\dot{\boldsymbol{p}}_{\boldsymbol{i}}$ and $\boldsymbol{\omega}_{\boldsymbol{i}}$

$$\dot{p}_i = \boldsymbol{J}_{P1}^{(i)} \dot{q}_1 + \cdots + \boldsymbol{J}_{Pi}^{(i)} \dot{q}_i = \boldsymbol{J}_P^{(i)} \dot{\mathbf{q}}, \qquad \forall i = \{1, ..., n_q\}, \tag{A.16}$$

$$\omega_i = \boldsymbol{J}_{O1}^{(i)} \dot{q}_1 + \cdots + \boldsymbol{J}_{Oi}^{(i)} \dot{q}_i = \boldsymbol{J}_O^{(i)} \dot{\mathbf{q}}, \qquad \forall i = \{1, ..., n_q\}, \tag{A.17}$$

where $\boldsymbol{J}_{Pi}^{(i)}$ corresponds to the i-th column of the positional Jacobian $\frac{\partial p}{\partial \mathbf{q}}$ and $\boldsymbol{J}_{Oi}^{(i)}$ corresponds to the i-th column of the orientation Jacobian $\frac{\partial \alpha}{\partial \mathbf{q}}$. Plugging in equations A.16 and A.17 in A.15, we get

$$\mathcal{T}_i = \frac{1}{2} m_i \dot{\mathbf{q}}^T \boldsymbol{J}_P^{iT} \boldsymbol{J}_P^i \dot{\mathbf{q}} + \frac{1}{2} \dot{\mathbf{q}}^T \boldsymbol{J}_O^{iT} R_i I_i R_i^T \boldsymbol{J}_O^i \dot{\mathbf{q}}, \qquad \forall i = \{1, ..., n_q\}, \tag{A.18}$$

Summing up over the n_q dynamic rigid bodies of the system, we get

$$\mathcal{T} = \frac{1}{2} \sum_{i=1}^{n_j} \sum_{j=1}^{n_j} b_{ij}(\mathbf{q}) \dot{q}_i \dot{q}_j = \frac{1}{2} \dot{\mathbf{q}}^T \boldsymbol{B}(\mathbf{q}) \dot{\mathbf{q}}, \tag{A.19}$$

where the matrix $\boldsymbol{B}(\mathbf{q})$ is the inertia matrix that can be written as

$$\boldsymbol{B}(\mathbf{q}) = \sum_{i=1}^{n_j} \left(m_i \boldsymbol{J}_P^{(i)T} \boldsymbol{J}_P^{(i)} + \boldsymbol{J}_O^{(i)T} \boldsymbol{R}_i \boldsymbol{I}_i^i \boldsymbol{R}_i^T \boldsymbol{J}_O^{(i)} \right). \tag{A.20}$$

Potential Energy Computation

Potential energy may be computed by summing up the contributions of each body. Assuming that the only conservative force that the manipulator is subjected to is the gravitational field of the Earth, thus neglecting possible elastic effects, the contribution of each body can be expressed as

$$\mathcal{U}_i = -m_i \boldsymbol{g}^T \boldsymbol{p}_{\boldsymbol{i}} \qquad \forall i = \{1, ..., n_q\}, \tag{A.21}$$

where $\boldsymbol{g}$ represents the gravity acceleration in vector form expressed in the global reference frame. Therefore, by simply summing up the contributions of each body, we obtain

$$\mathcal{U} = -\sum_{i=1}^{n_j} m_i \boldsymbol{g}^T \boldsymbol{p}_{\boldsymbol{i}}. \tag{A.22}$$

Equations of Motion

Having calculated the potential and kinetic energies, we can write the Lagrangian of the system as

$$\mathcal{L} = \mathcal{T} - \mathcal{U} = \frac{1}{2}\sum_{i=1}^{n_j}\sum_{j=1}^{n_j} b_{ij}(\mathbf{q})\dot{q}_i\dot{q}_j + \sum_{i=1}^{n_j}\left(m_i \boldsymbol{g}^T \boldsymbol{p}_i(\mathbf{q})\right) \tag{A.23}$$

Then, we can plug in equation A.23 into A.14 and obtain:

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) + \frac{\partial \mathcal{L}}{\partial q_i} = &\sum_{j=1}^{n_j} b_{ij}(\mathbf{q})\ddot{q}_j + \sum_{j=1}^{n_j}\frac{db_{ij}(\mathbf{q})}{dt}\dot{q}_j + \sum_{j=1}^{n_j} b_{ij}(\mathbf{q})\ddot{q}_j + \sum_{j=1}^{n_j}\sum_{k=1}^{n_j}\frac{\partial b_{ij}(\mathbf{q})}{\partial q_k}\dot{q}_k\dot{q}_j \\ &+ \frac{1}{2}\sum_{j=1}^{n_j}\sum_{k=1}^{n_j}\frac{\partial b_{jk}(\mathbf{q})}{\partial q_i}\dot{q}_k\dot{q}_j - \sum_{j=1}^{n_j}\left(m_j \boldsymbol{g}^T \boldsymbol{J}_{Pi}^{(j)}(\mathbf{q})\right) \end{aligned} \tag{A.24}$$

As such, we may define the last term of equation A.24 as a special function that only depends on the position of the coordinates $\mathbf{q}$ and not on their velocity, called the *gravitational term*

$$g_i(\mathbf{q}) = -\sum_{j=1}^{n_j}\left(m_j \boldsymbol{g}^T \boldsymbol{J}_{Pi}^{(j)}(\mathbf{q})\right) \tag{A.25}$$

Also, we can define a convenient operator h_{ijk} over the elements of the Inertia Matrix b_{ij} to further simplify the equations. The resulting coefficients can be written as

$$h_{ijk} = \frac{\partial b_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial b_{jk}}{\partial q_i} \tag{A.26}$$

Using equations A.25 and A.26 in combination with A.24, we obtain the general equations of motion

$$\sum_{j=1}^{n_j} b_{ij}(\mathbf{q})\ddot{q}_j + \sum_{j=1}^{n_j}\sum_{k=1}^{n_j} h_{ijk}(\mathbf{q})\dot{q}_k\dot{q}_j + g_i(\mathbf{q}) = f_i \qquad \forall i = \{1, ..., n_q\} \tag{A.27}$$

These equations can be further simplified by expressing them in vector form as

$$\boldsymbol{B}(\mathbf{q})\ddot{\mathbf{q}} + \boldsymbol{C}(\mathbf{q},\dot{\mathbf{q}}) + \boldsymbol{F}_v\dot{\mathbf{q}} + \boldsymbol{F}_s\,\mathrm{sgn}(\dot{\mathbf{q}}) + \boldsymbol{g}(\mathbf{q}) = \boldsymbol{\tau} - \boldsymbol{J}^T\boldsymbol{h} \tag{A.28}$$

In the previous equations, we introduced the term $\boldsymbol{F}_v\dot{\mathbf{q}}$, which accounts for *viscous friction* by a simple proportional relationship with the velocity of the coordinates. Note that $\boldsymbol{F}_v$ is a constant diagonal matrix. We also introduced the *static friction* term $\boldsymbol{F}_s\,\mathrm{sgn}(\dot{\mathbf{q}})$, which also considers a diagonal matrix $\boldsymbol{F}_s$ to model a decoupled effect. Another added term is the torque disturbance $\boldsymbol{J}^T\boldsymbol{h}$, which accounts for the effect of a payload on the final joint through the geometrical Jacobian $\begin{bmatrix} J_P & J_\omega \end{bmatrix}^T$ and external disturbance h. Finally, the matrix $\boldsymbol{C}(\mathbf{q},\dot{\mathbf{q}})$ accounts for the Coriolis and centrifugal effects, and its choice is not unique. Elements of this matrix, c_{ij}, can be constructed as

$$c_{ij} = \sum_{k=1}^{n_j} c_{ijk} \dot{q}_k \tag{A.29}$$

where the coefficients c_{ijk} are called the *Christoffel Symbols* of the first kind

$$c_{ijk} = \frac{1}{2}\left(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} - \frac{\partial b_{jk}}{\partial q_i}\right) \tag{A.30}$$

In this way, the matrix $\dot{\boldsymbol{B}}(\mathbf{q}) - 2\boldsymbol{C}(\mathbf{q}, \dot{\mathbf{q}})$ is be *skew symmetric*, a property that proves useful while designing a model-based control strategies. In summary, the components of (A.28):

$\mathbf{B}(\mathbf{q})$	$\in \mathbb{R}^{n_q \times n_q}$	Generalized mass matrix (orthogonal).
$\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$	$\in \mathbb{R}^{n_q}$	Generalized position, velocity, and acceleration vectors.
$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$	$\in \mathbb{R}^{n_q}$	Coriolis and centrifugal terms.
$\mathbf{F_v}$	$\in \mathbb{R}^{n_q \times n_q}$	Viscous friction.
$\mathbf{F_s}$	$\in \mathbb{R}^{n_q \times n_q}$	Static friction.
$\mathbf{g}(\mathbf{q})$	$\in \mathbb{R}^{n_q}$	Gravitational terms.
$\boldsymbol{\tau}$	$\in \mathbb{R}^{n_q}$	External generalized forces.
$\mathbf{h}$	$\in \mathbb{R}^{n_c}$	External Cartesian forces (e.g. from contacts).
$\mathbf{J}$	$\in \mathbb{R}^{n_c \times n_q}$	Geometric Jacobian corresponding to the external forces.

A.3 Estimators

Estimators try to estimate unmeasured variables (often internal system states) by exploiting knowledge about the dynamics of the system at hand together with the input and output signals as can be seen in Fig. A.2.

An LTI systems Luenberger observer of the form

$$\hat{\xi}(t+1) = A\hat{\xi}(t) + B\tilde{u}(t) + L\Big(\tilde{y}(t) - C\hat{\xi}(t)\Big), \tag{A.31a}$$

$$\hat{y}(t) = C\hat{\xi}(t) + D\tilde{u}(t), \tag{A.31b}$$

is often used where ξ is the system state estimate, $\hat{y}$ is the output estimate, $\tilde{u}$ is the measured system input, $\tilde{y}$ is the measured output, A, B, C, D are the model matrices, and L is the gain of the observer that causes the estimation error $\xi(t) - \hat{\xi}(t) \to 0$ asymptotically assuming measurement noise has a zero mean and no model mismatch as well as matrices A, C satisfy observability condition.

This kind of observers is limited to LTI systems and cannot deal with nonlinear dynamics and constraints.

A.3.1 Kalman Filter

Kalman Filter (KF) is another widely used filter. It tries to minimize the variance of error estimation in a stochastic framework [33]. Implementing an EKF allows

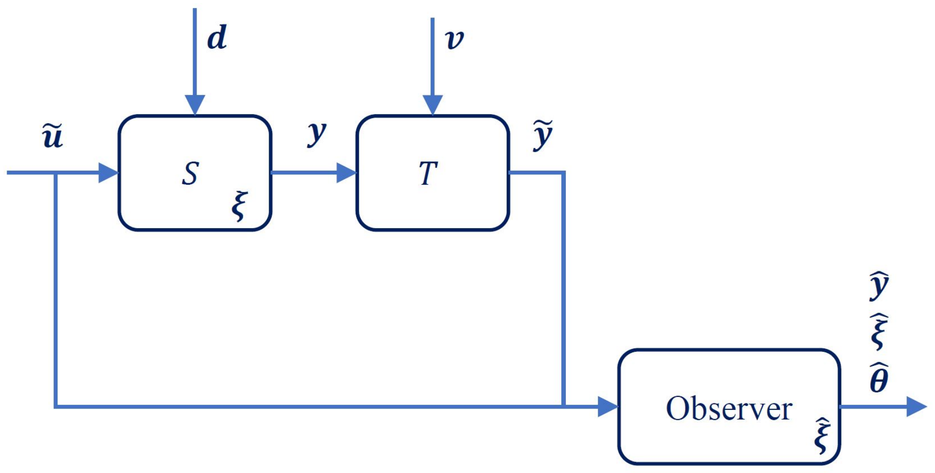


Figure A.2: Sample filtering scheme where the observer exploits knowledge about the system dynamics S to estimate output measurements $\hat{y}$, state variables $\hat{\xi}$, and perhaps unknown parameters θ. The system takes measured input $\tilde{u}$ and is subject to noise d. System outputs commands y to a transducer T, which get noised by process disturbance v, producing $\tilde{y}$.

estimation for nonlinear systems, still without the possibility of easily including constraints. A sample linear KF for position tracking[1] has state vector

$$x = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z})^T, \tag{A.32}$$

containing positional data and the accompanying first and second derivatives. It has a process model (dynamics) of the form

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}, \tag{A.33}$$

where A is the state transition matrix, Bu is control input, and w is the process noise with covariance Q.

For a simple process model with no input, A has the form

$$\begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}(\Delta t)^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}(\Delta t)^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{A.34}$$

Lastly, the measurement model as a function of the state has the form $z_k =$

[1]http://campar.in.tum.de/Chair/KalmanFilter

$Hx_k + v_k$ below

$$\begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \\ z_k \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} + v_k. \tag{A.35}$$

where v_k is the measurement noise with covariance R.

The KF update cycle has two steps. Step 1 is the prediction step where we use (A.33) to get a a-priori estimate $\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$ and a covariance estimate $P_k^- = AP_{k-1}A^T + Q$. The second step is a measurement update step that corrects our estimate. We compute the Kalman gain first $K_k = P_k^- H^T \Big(HP_k^- H^T + R \Big)^{-1}$, then we update the estimate $\hat{x}_k = \hat{x}_k^- + K_k \Big(z_k - H\hat{x}_k^- \Big)$, and finally we update the covariance $P_k = \Big(I - K_k H \Big) P_k^-$.

In is noteworthy that Q and R are tunable. Larger Q means that the KF gives more attention to changes in data. Larger R means that the KF considers the measurements as very noisy ad follows them less closely.

A.3.2 Moving Horizon Estimator

MHE solves the problems that LTI estimators face that we stated previously. An MHE is a semi-brute-force numerical optimization technique that tries to estimate variables that would produce, using the knowledge about the system, the nearest results to the measured ones. It solves a problem in real-time of the following form

$$\min_{\hat{\xi}(-M|t),\hat{d}(-M|t),\dots,\hat{d}(0|t)} \sum_{i=0}^{M} \|W(\tilde{y}(t-i) - \hat{y}(-i \mid t)\|_2^2 \tag{A.36a}$$

subject to

$$\hat{\xi}(i+1 \mid t) = f_\xi\Big(\hat{\xi}(i \mid t), \tilde{u}(t+i), \hat{d}(i \mid t), \theta\Big), i = -M, \dots, -1 \tag{A.36b}$$

$$\hat{y}(i \mid t) = g_\xi\Big(\hat{\xi}(i \mid t), \tilde{u}(t+i), \hat{d}(i \mid t), 0, \theta\Big), i = -M, \dots, 0 \tag{A.36c}$$

$$h_\xi\Big(\hat{\xi}(i \mid t)\Big) \geq 0, i = -M, \dots, 0, \tag{A.36d}$$

where W $\in \mathbb{R}^{n_y \times n_y} = diag(w_1, \dots, w_{n_y})$ is a weighting matrix, and the 0 argument in (A.36c) stands for an assumed measurements noise of 0.

Basically, it estimates variables for step t-M, meaning M steps in the past, then propagates the solution via numerical integration of the model. This results in an estimate for the variables at current time. At the next step, an MHE repeats the same process all over.

A.4 A Fast Visibility Check Using Horizon Culling

Based on Fig. 1.8, the following equations are used to obtain the visibility score which is used to handle occlusions in [15],

$$r_{ch_i} = p_{h_i} - p_c, \tag{A.37}$$
$$r_{ct_j} = p_{t_j} - p_c, \tag{A.38}$$
$$p_{proj} = r_{ch_i}^T r_{ct_j}, \tag{A.39}$$

where r_{ch_i} is the vector to the target, r_{ct_j}, r_{ct_j} is the vector to the center of ellipsoid S_{σ_s} and p_{proj} is the component of r_{ch_i} in direction of r_{ct_j}.

Basically, if $p_{proj} > r_{ct_j}^T r_{ct_j} - 1$ and angle $\alpha <$ angle β, then the point being checked is occluded.

A visibility score is directly evaluated in order to avoid trigonometric functions,

$$d_v = \frac{p_{proj}}{r_{ch_i}^T r_{ch_i}} > r_{ct_j}^T r_{ct_j}. \tag{A.40}$$

The visibility cost is defined as $c_{vis}(x)$:

$$c_{vis}(x) = \begin{cases} \|d_v\|_{Q_v} & \text{if } d_v > 0 \text{ and } p_{proj} > r_{ct_j}^T r_{ct_j} - 1 \\ 0 & \text{otherwise.} \end{cases} \tag{A.41}$$

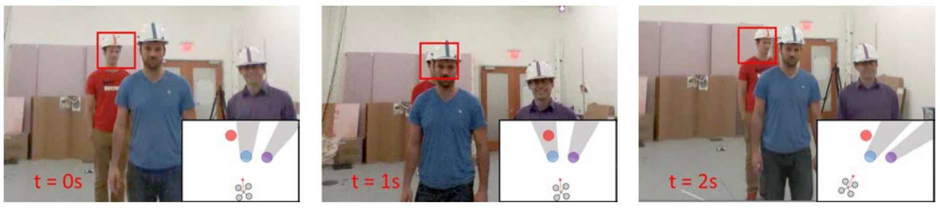


Figure A.3: Occlusion cones produced by occluding obstacles are estimated using a fast visibility check based on horizon culling [15].

Appendix B

Graph Appendix

B.1 Deterministic Hard Shadows and a Gradient-Based Solution

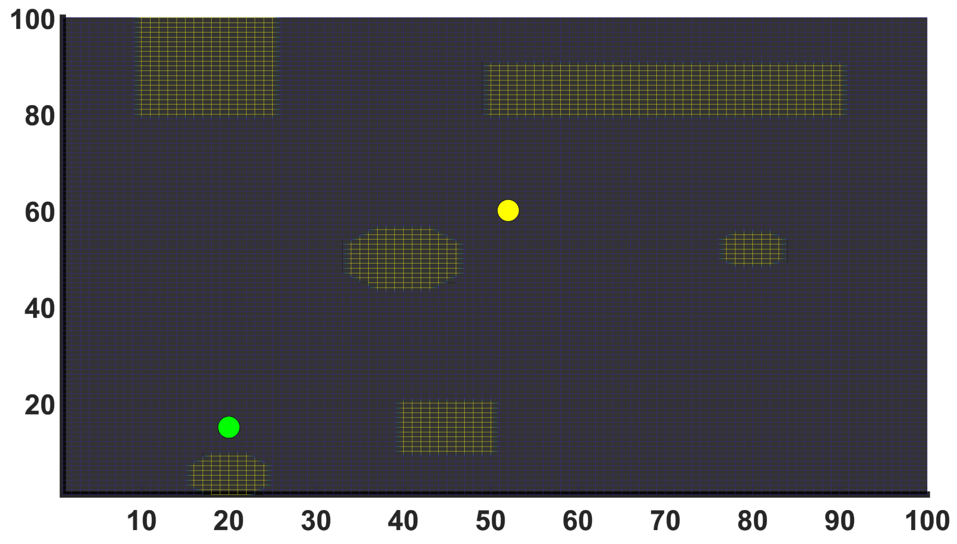


Figure B.1: Topview of a 2D environment containing obstacles with yellow face color. Yellow ball is the light position, green ball is the robot position.

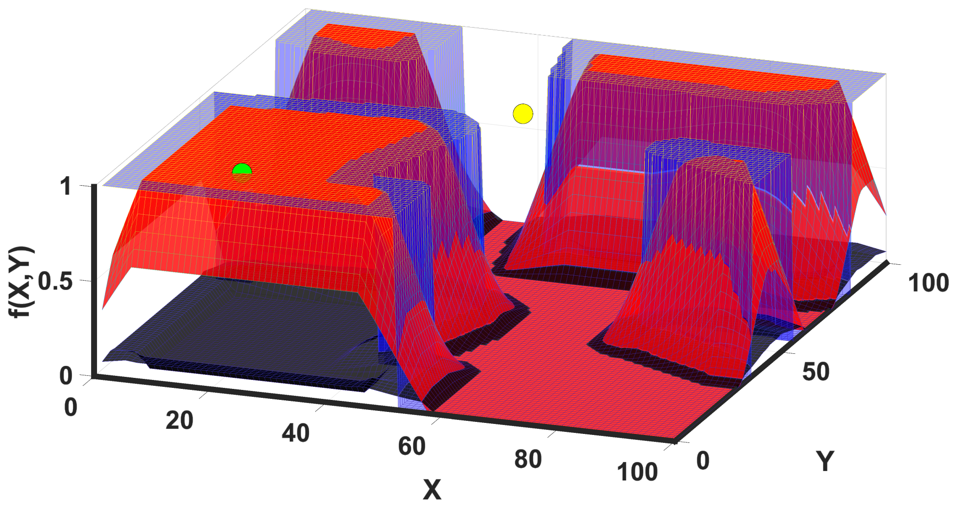


Figure B.2: Sideview of the environment in Fig. B.1 showing the deterministic *shadow field* mesh in blue, a Gaussian filter convolution of *shadow field* in red, and the gradient of the Gaussian filter convolution in black.

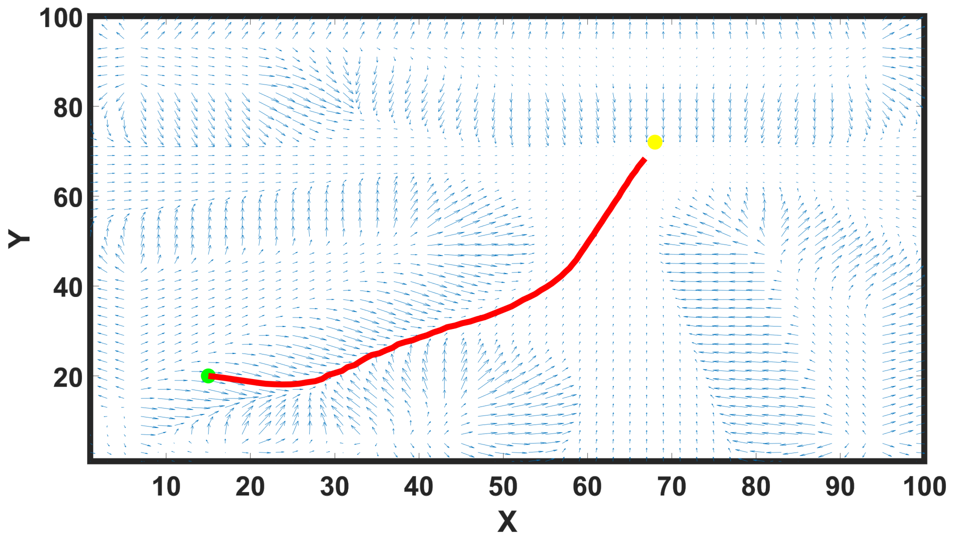


Figure B.3: Quiver plot of the resulting *shadow field* from Fig. B.2 as well as a highlighted path resulting from gradient-base planner, specifically a planner using a naive gradient-descent algorithm.

B.2 More Kinova and ALMA Simulations

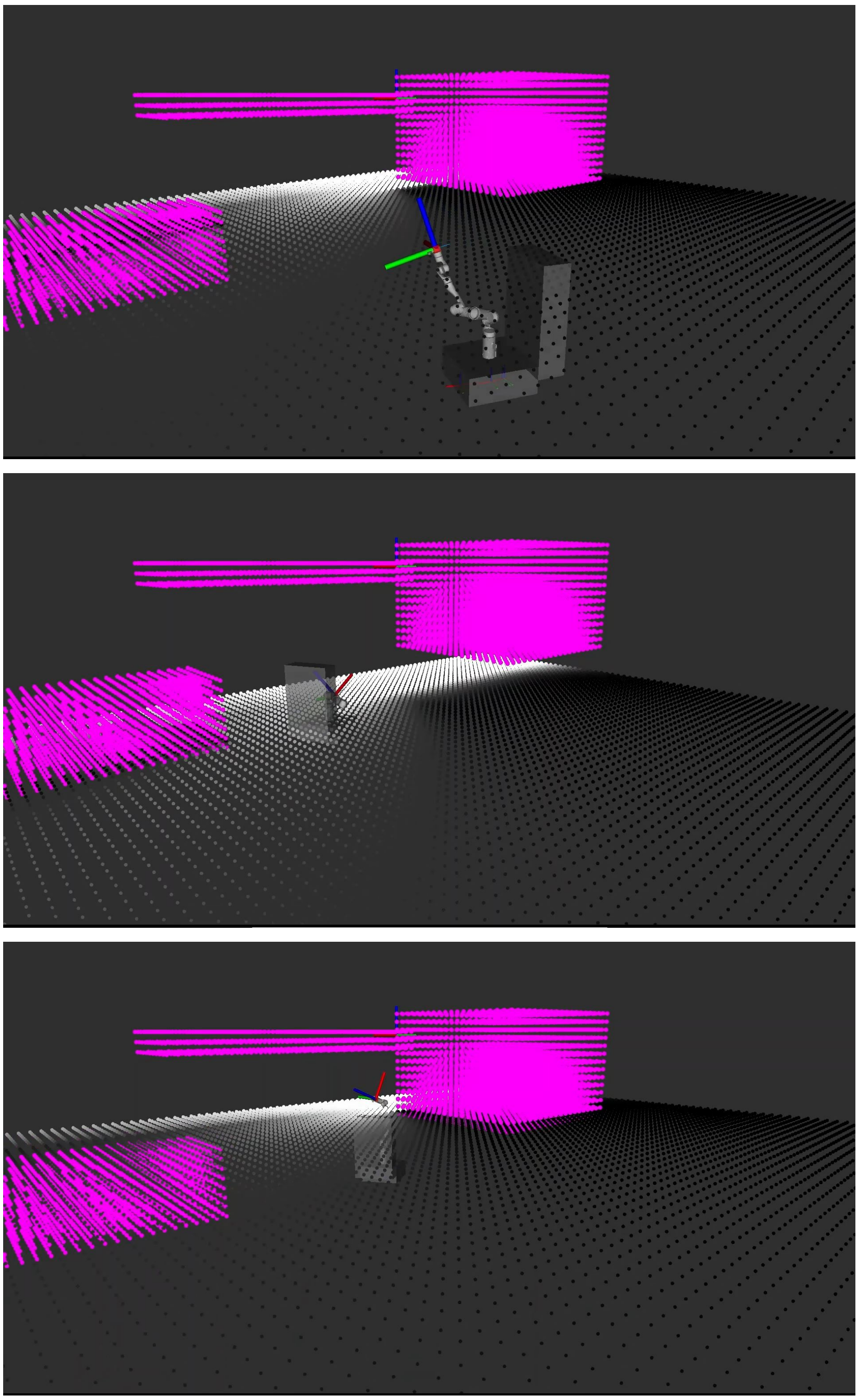


Figure B.4: Sideview of Kinova 7DOF mobile manipulator evolution in simulation. The end-effector finds its way from its invisible initial position to position with 100% visibility. We show the *shadow field* slice at end-effector level.

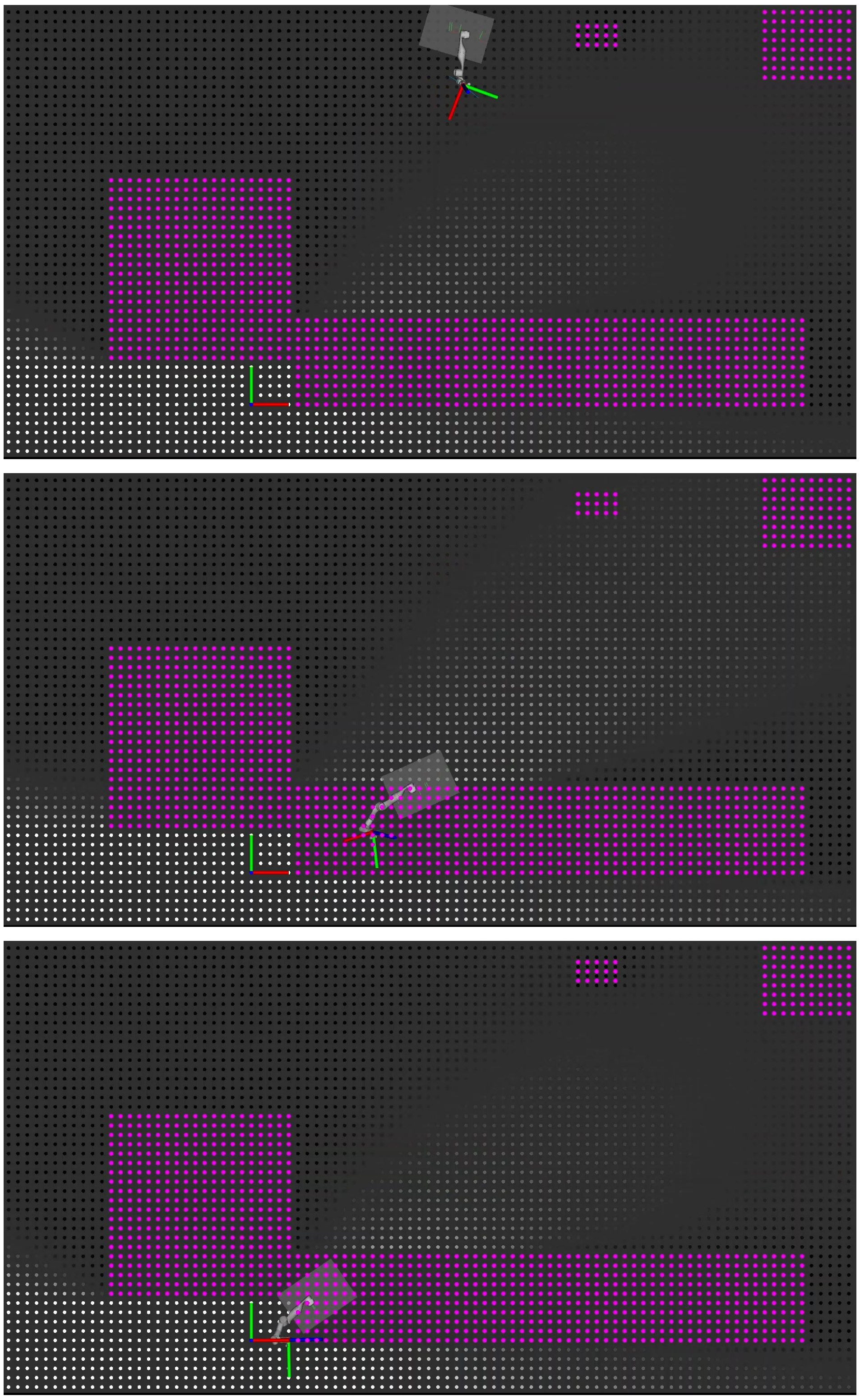


Figure B.5: Topview of Kinova 7DOF mobile manipulator evolution in simulation. The end-effector finds its way from its invisible initial position to position with 100% visibility. We show the *shadow field* slice at end-effector level.

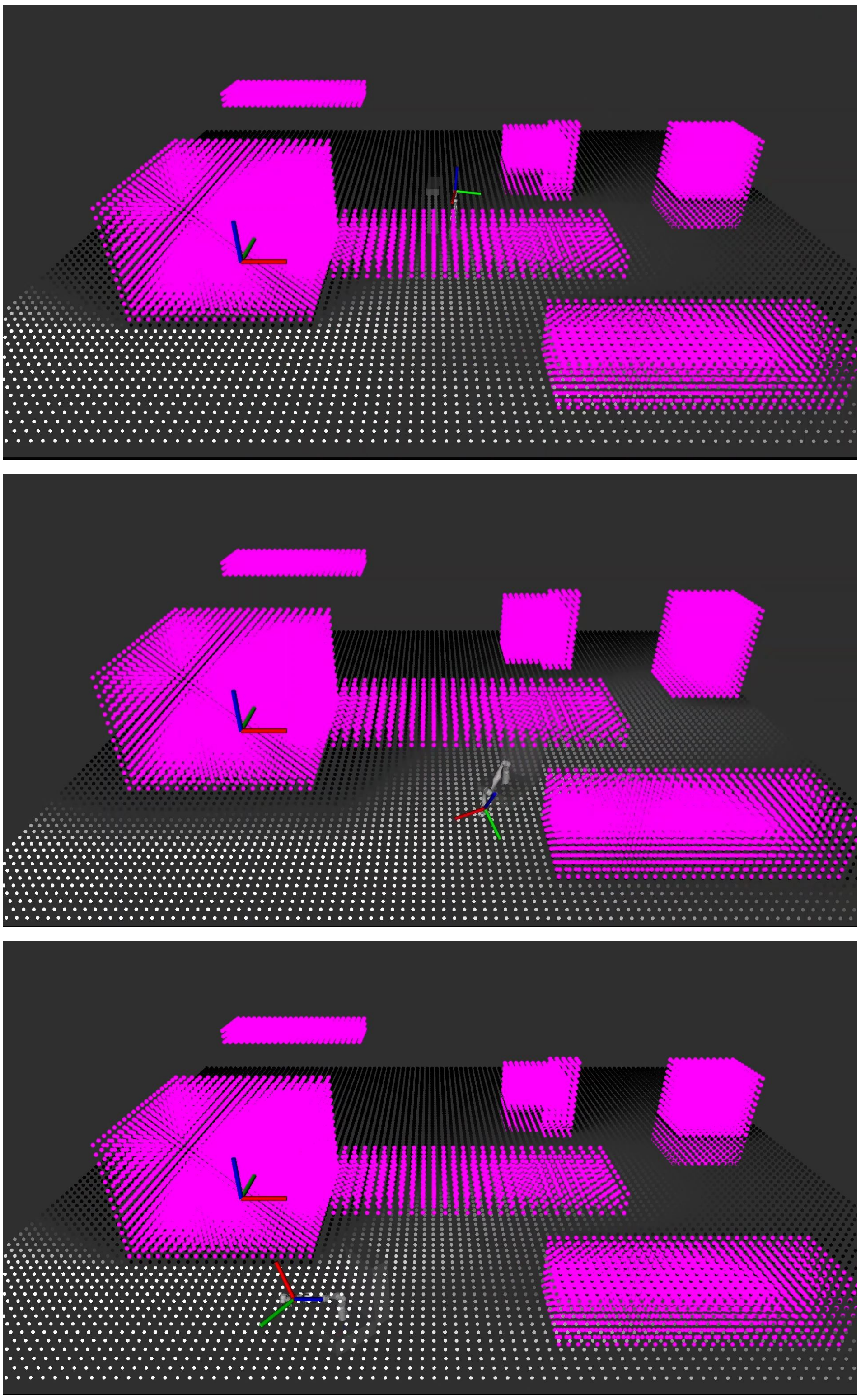


Figure B.6: Frontview of Kinova 7DOF mobile manipulator evolution in simulation. The end-effector finds its way from its invisible initial position to position with 100% visibility. We show the *shadow field* slice at end-effector level.

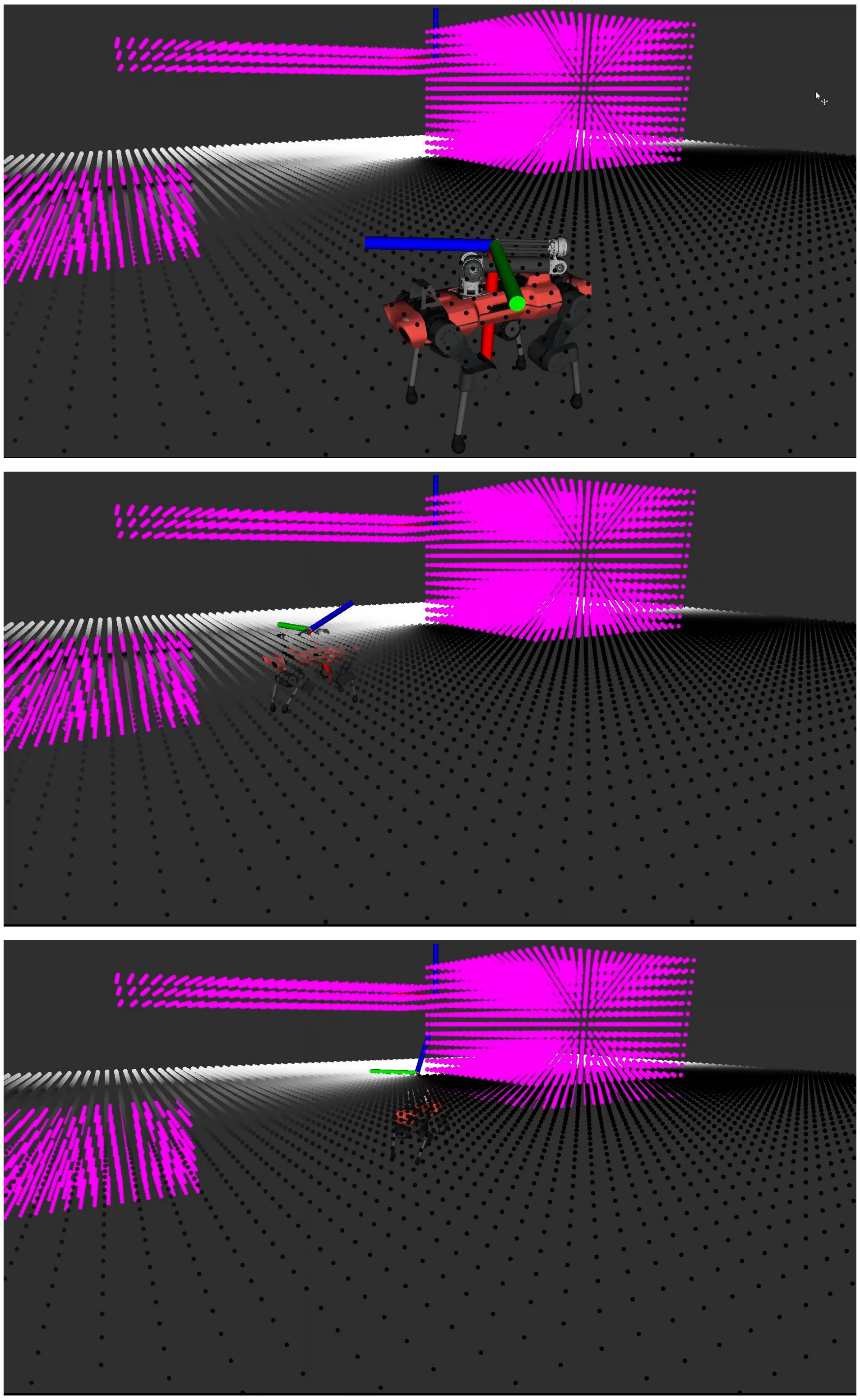


Figure B.7: Sideview of ALMA evolution in simulation. The end-effector finds its way from its invisible initial position to position with 100% visibility. We show the *shadow field* slice at end-effector level.

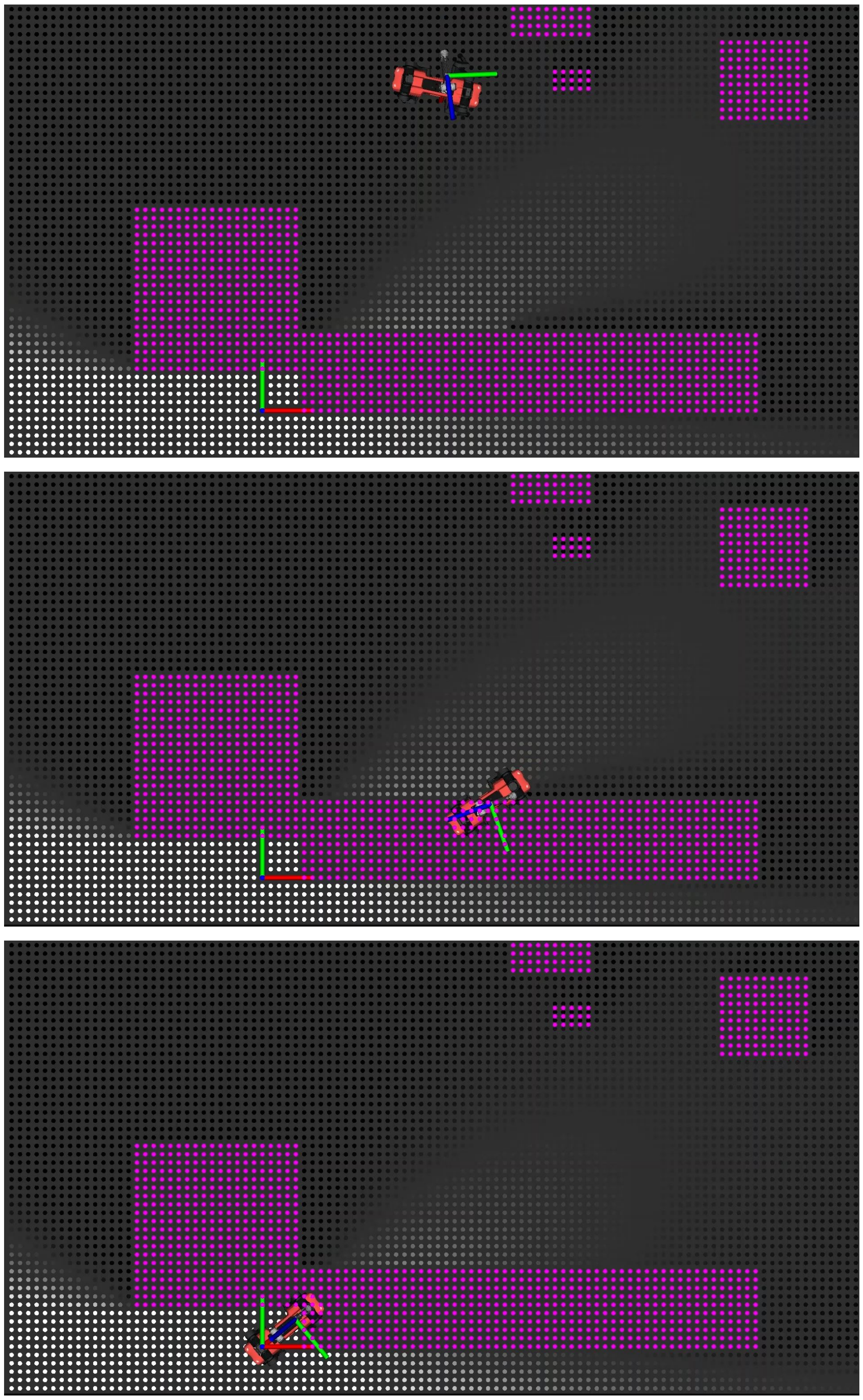


Figure B.8: Topview of ALMA evolution in simulation. The end-effector finds its way from its invisible initial position to position with 100% visibility. We show the *shadow field* slice at end-effector level.

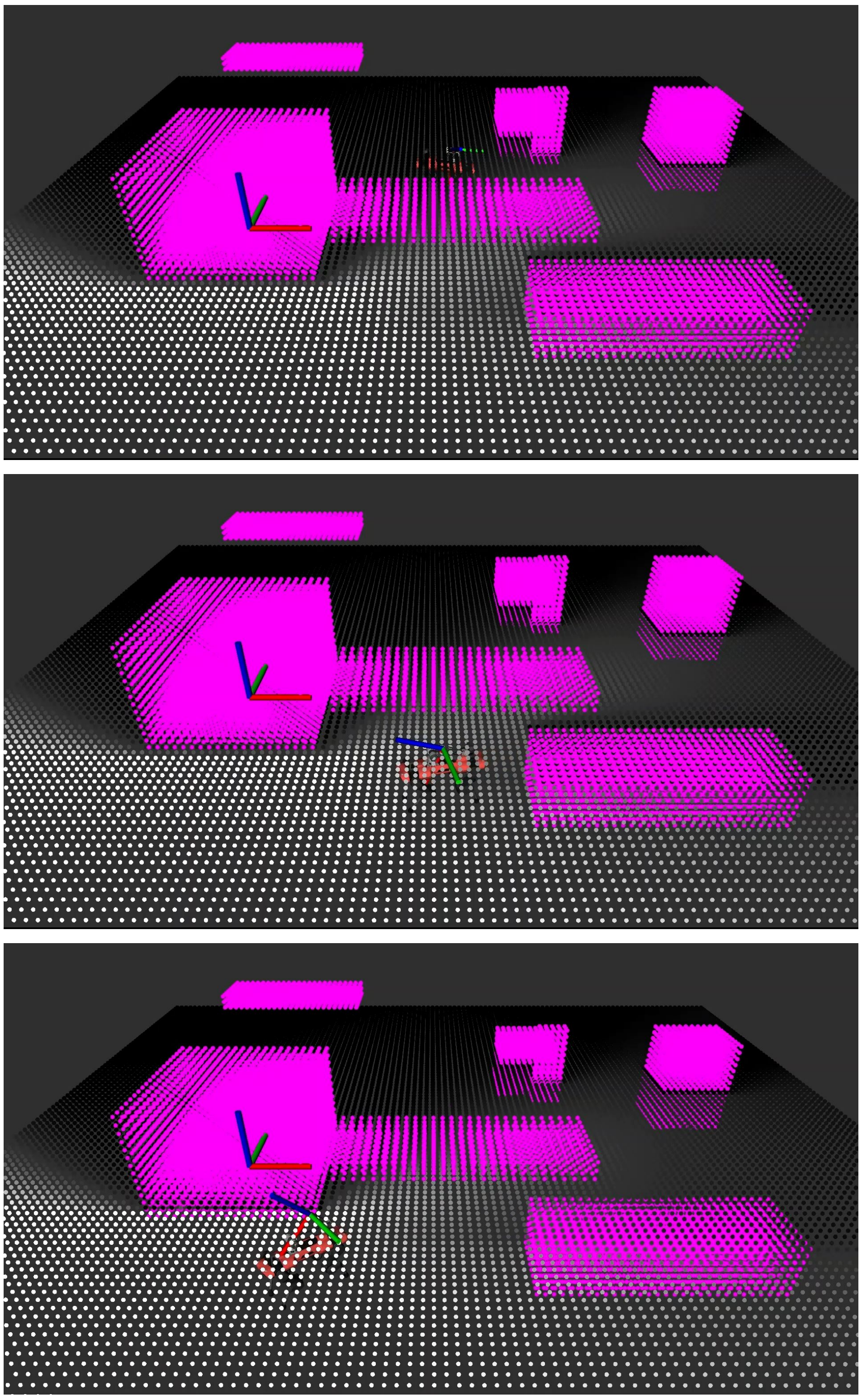


Figure B.9: Frontview of ALMA evolution in simulation. The end-effector finds its way from its invisible initial position to position with 100% visibility. We show the *shadow field* slice at end-effector level.

Appendix C

Datasheets

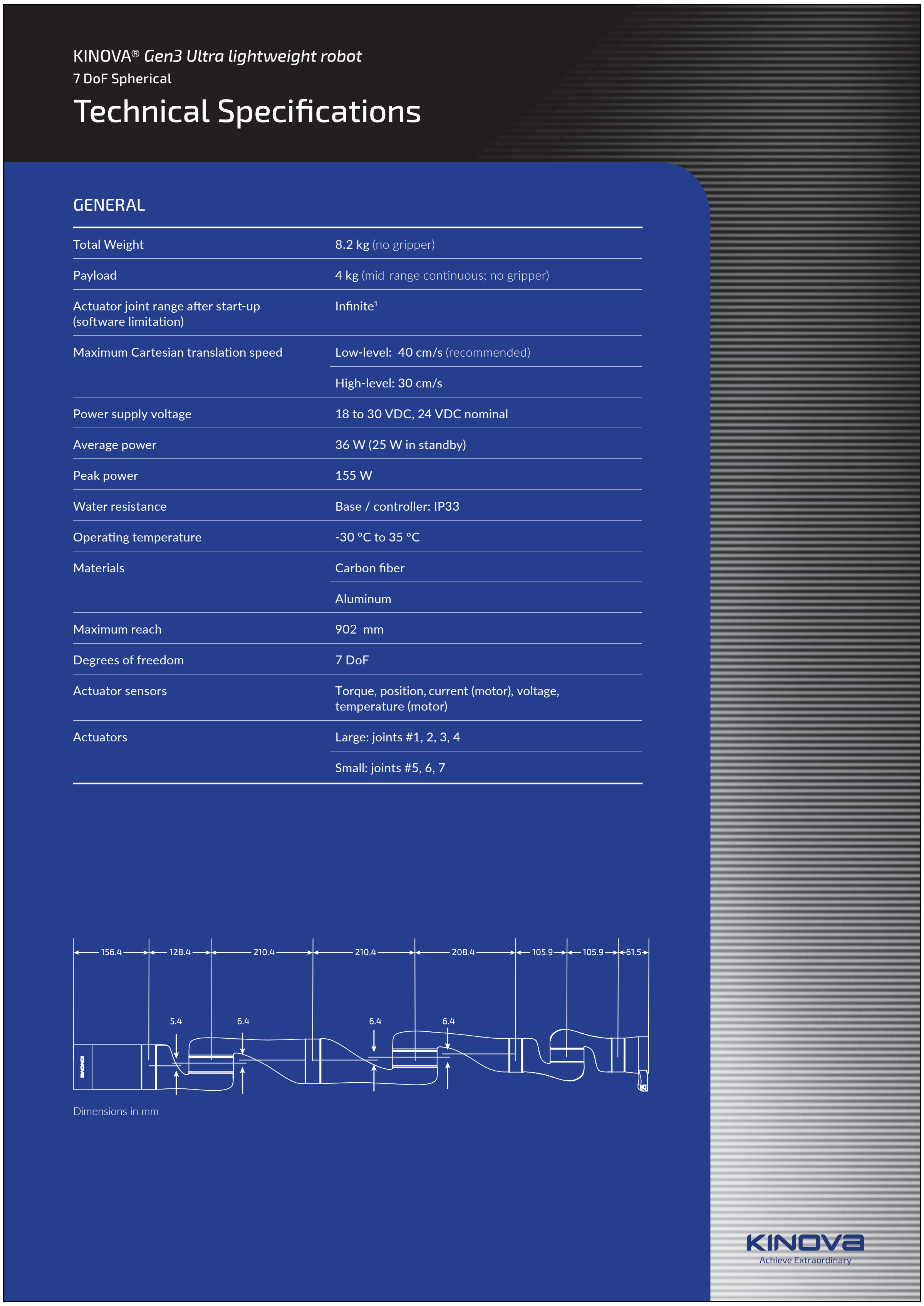
KINOVA® *Gen3 Ultra lightweight robot*

7 DoF Spherical

Technical Specifications

KINOVA

Achieve Extraordinary



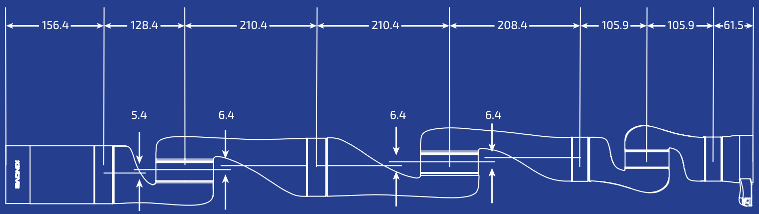
KINOVA® *Gen3 Ultra lightweight robot*

7 DoF Spherical

Technical Specifications

GENERAL

Total Weight	8.2 kg (no gripper)
Payload	4 kg (mid-range continuous; no gripper)
Actuator joint range after start-up (software limitation)	Infinite[1]
Maximum Cartesian translation speed	Low-level: 40 cm/s (recommended)
	High-level: 30 cm/s
Power supply voltage	18 to 30 VDC, 24 VDC nominal
Average power	36 W (25 W in standby)
Peak power	155 W
Water resistance	Base / controller: IP33
Operating temperature	-30 °C to 35 °C
Materials	Carbon fiber
	Aluminum
Maximum reach	902 mm
Degrees of freedom	7 DoF
Actuator sensors	Torque, position, current (motor), voltage, temperature (motor)
Actuators	Large: joints #1, 2, 3, 4
	Small: joints #5, 6, 7



Dimensions in mm

KINOVA
Achieve Extraordinary

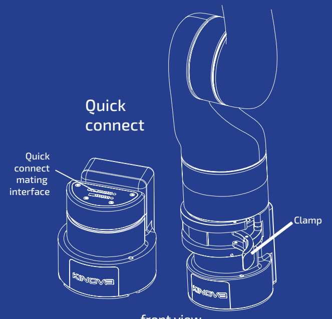
KINOVA® *Gen3 Ultra lightweight robot*

7 DoF Spherical

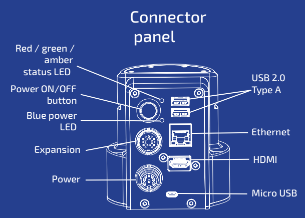
Technical Specifications

BASE CONTROLLER

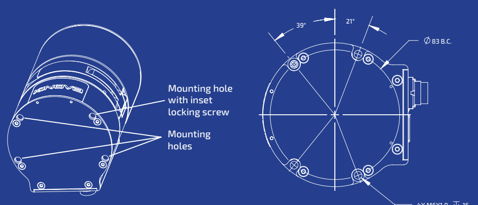
Software	KINOVA® KORTEX™	
Internal communications	2 x Fast Ethernet (100 Mbps)	
API compatibility	Windows 10, Linux Ubuntu 16.04, ROS Kinetic	
Programming languages	C++, Python, MATLAB[2]	
Supported web browser	Google Chrome 64+	
Controller interfaces	USB	Gamepad
	Ethernet (Web App + API)	1 Gbps
	HDMI	1.4a[3]
	Wi-Fi (Web App + API)	IEEE 802.11a/b/g/n
	Bluetooth 4.0 + LE[3]	
	Digital I/O[3]	
Control system frequency	1 kHz	
Servoing modes	High-level, low-level	
Low-level control	Position, velocity, current[3], torque[3]	
High-level control	Cartesian position/velocity, joint position/velocity, force[3], torque[3]	
Controller sensors	Voltage, current, accelerometer, temperature and gyroscope	



front view

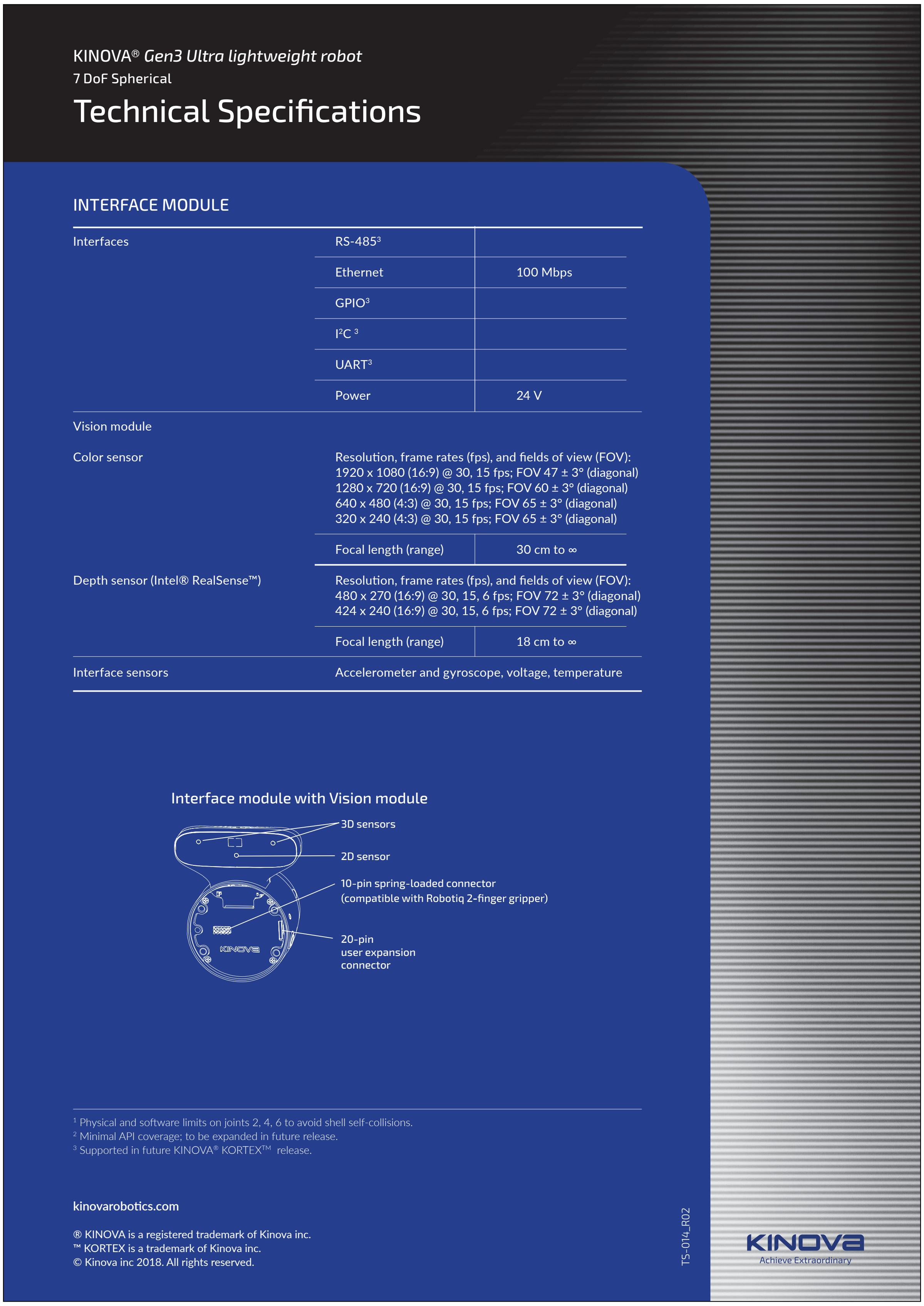


rear view



bottom view

KINOVA
Achieve Extraordinary



KINOVA® *Gen3 Ultra lightweight robot*

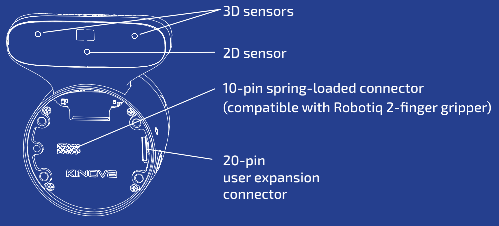
7 DoF Spherical

Technical Specifications

INTERFACE MODULE

Interfaces	RS-485[3]	
	Ethernet	100 Mbps
	GPIO[3]	
	I^2C [3]	
	UART[3]	
	Power	24 V
Vision module		
Color sensor	Resolution, frame rates (fps), and fields of view (FOV): 1920 x 1080 (16:9) @ 30, 15 fps; FOV 47 ± 3° (diagonal) 1280 x 720 (16:9) @ 30, 15 fps; FOV 60 ± 3° (diagonal) 640 x 480 (4:3) @ 30, 15 fps; FOV 65 ± 3° (diagonal) 320 x 240 (4:3) @ 30, 15 fps; FOV 65 ± 3° (diagonal)	
	Focal length (range)	30 cm to ∞
Depth sensor (Intel® RealSense™)	Resolution, frame rates (fps), and fields of view (FOV): 480 x 270 (16:9) @ 30, 15, 6 fps; FOV 72 ± 3° (diagonal) 424 x 240 (16:9) @ 30, 15, 6 fps; FOV 72 ± 3° (diagonal)	
	Focal length (range)	18 cm to ∞
Interface sensors	Accelerometer and gyroscope, voltage, temperature	

Interface module with Vision module

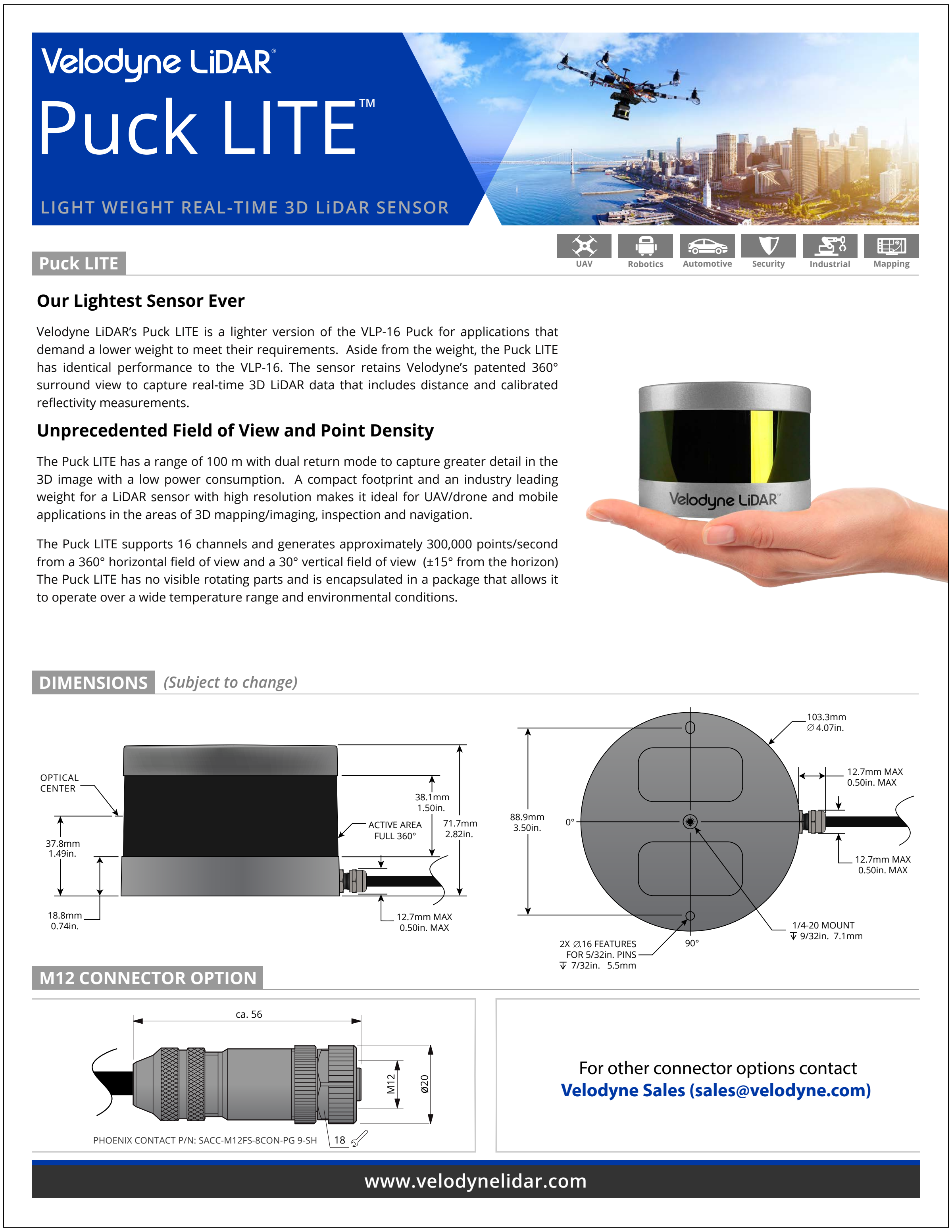

3D sensors
2D sensor
10-pin spring-loaded connector
(compatible with Robotiq 2-finger gripper)
20-pin
user expansion
connector
KINOVA

[1] Physical and software limits on joints 2, 4, 6 to avoid shell self-collisions.
[2] Minimal API coverage; to be expanded in future release.
[3] Supported in future KINOVA® KORTEX™ release.

kinovarobotics.com

TS-014_R02

KINOVA
Achieve Extraordinary



Velodyne LiDAR®

Puck LITE™

LIGHT WEIGHT REAL-TIME 3D LiDAR SENSOR

UAV
Robotics
Automotive
Security
Industrial
Mapping

Puck LITE

Our Lightest Sensor Ever

Velodyne LiDAR's Puck LITE is a lighter version of the VLP-16 Puck for applications that demand a lower weight to meet their requirements. Aside from the weight, the Puck LITE has identical performance to the VLP-16. The sensor retains Velodyne's patented 360° surround view to capture real-time 3D LiDAR data that includes distance and calibrated reflectivity measurements.

Unprecedented Field of View and Point Density

The Puck LITE has a range of 100 m with dual return mode to capture greater detail in the 3D image with a low power consumption. A compact footprint and an industry leading weight for a LiDAR sensor with high resolution makes it ideal for UAV/drone and mobile applications in the areas of 3D mapping/imaging, inspection and navigation.

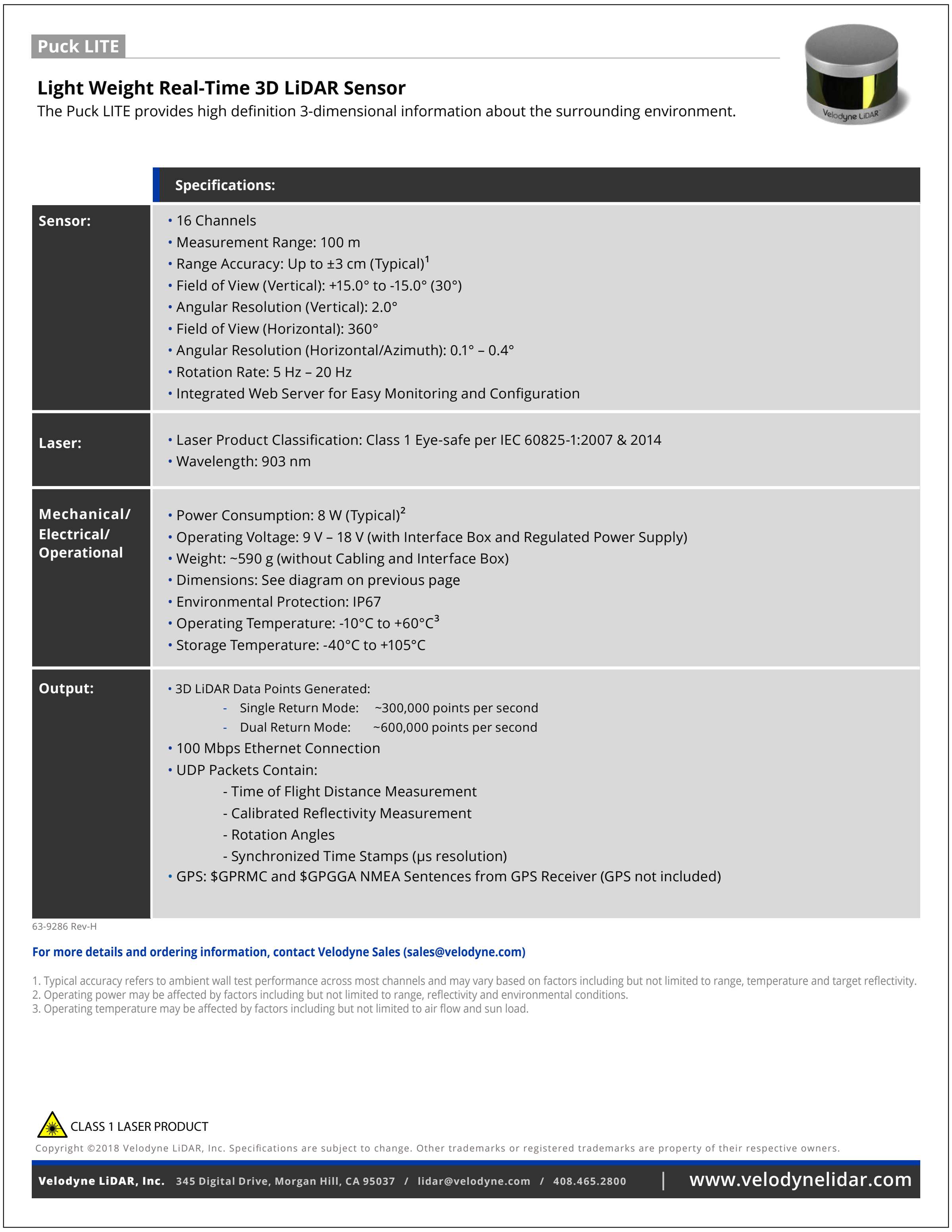
The Puck LITE supports 16 channels and generates approximately 300,000 points/second from a 360° horizontal field of view and a 30° vertical field of view (±15° from the horizon) The Puck LITE has no visible rotating parts and is encapsulated in a package that allows it to operate over a wide temperature range and environmental conditions.

DIMENSIONS *(Subject to change)*

M12 CONNECTOR OPTION

For other connector options contact
Velodyne Sales (sales@velodyne.com)

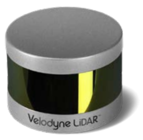
www.velodynelidar.com



Puck LITE

Light Weight Real-Time 3D LiDAR Sensor

The Puck LITE provides high definition 3-dimensional information about the surrounding environment.



	Specifications:
Sensor:	• 16 Channels • Measurement Range: 100 m • Range Accuracy: Up to ±3 cm (Typical)[1] • Field of View (Vertical): +15.0° to -15.0° (30°) • Angular Resolution (Vertical): 2.0° • Field of View (Horizontal): 360° • Angular Resolution (Horizontal/Azimuth): 0.1° – 0.4° • Rotation Rate: 5 Hz – 20 Hz • Integrated Web Server for Easy Monitoring and Configuration
Laser:	• Laser Product Classification: Class 1 Eye-safe per IEC 60825-1:2007 & 2014 • Wavelength: 903 nm
Mechanical/ Electrical/ Operational	• Power Consumption: 8 W (Typical)[2] • Operating Voltage: 9 V – 18 V (with Interface Box and Regulated Power Supply) • Weight: ~590 g (without Cabling and Interface Box) • Dimensions: See diagram on previous page • Environmental Protection: IP67 • Operating Temperature: -10°C to +60°C[3] • Storage Temperature: -40°C to +105°C
Output:	• 3D LiDAR Data Points Generated: - Single Return Mode: ~300,000 points per second - Dual Return Mode: ~600,000 points per second • 100 Mbps Ethernet Connection • UDP Packets Contain: - Time of Flight Distance Measurement - Calibrated Reflectivity Measurement - Rotation Angles - Synchronized Time Stamps (μs resolution) • GPS: $GPRMC and $GPGGA NMEA Sentences from GPS Receiver (GPS not included)

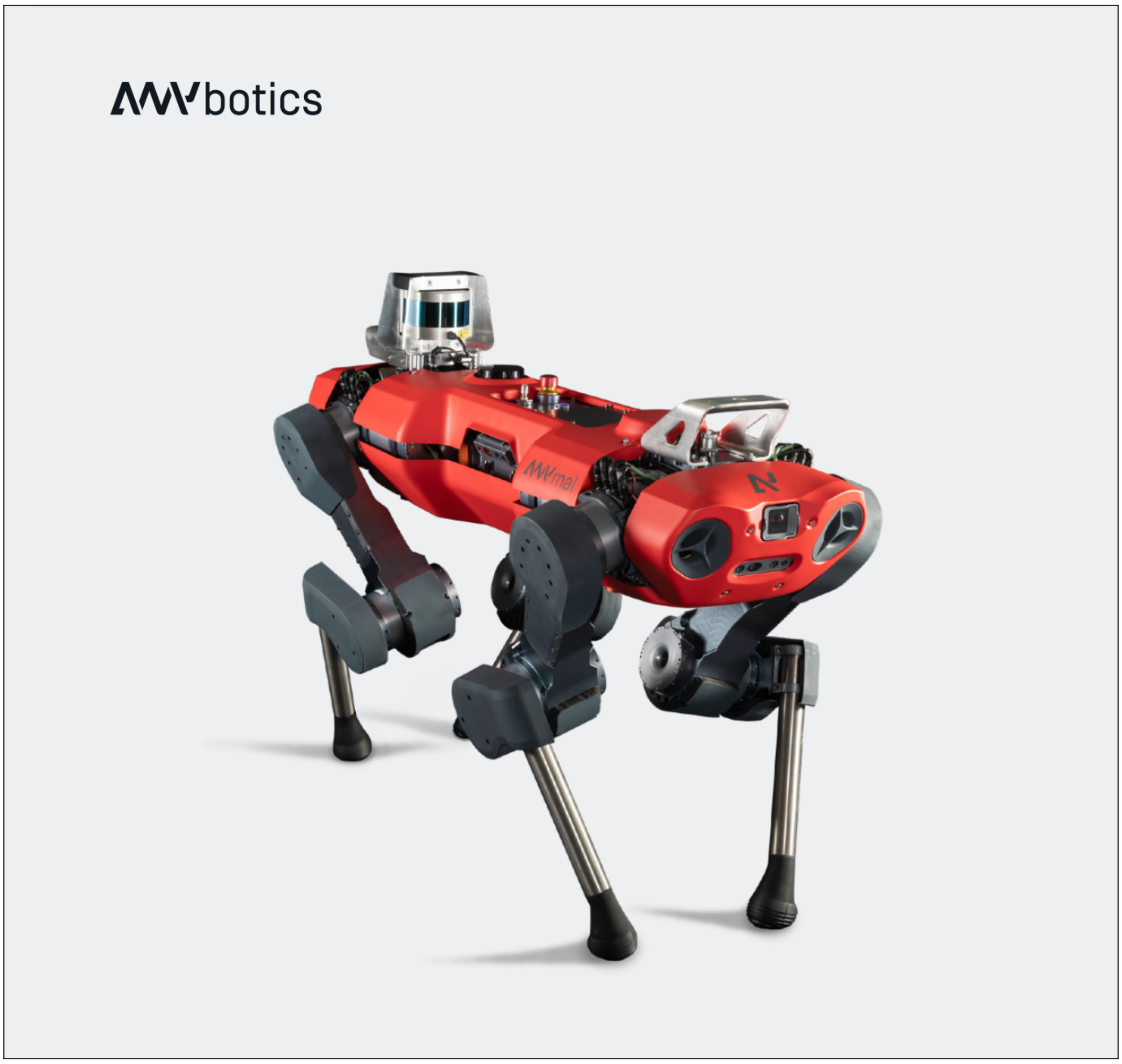
63-9286 Rev-H

For more details and ordering information, contact Velodyne Sales (sales@velodyne.com)

1. Typical accuracy refers to ambient wall test performance across most channels and may vary based on factors including but not limited to range, temperature and target reflectivity.
2. Operating power may be affected by factors including but not limited to range, reflectivity and environmental conditions.
3. Operating temperature may be affected by factors including but not limited to air flow and sun load.

CLASS 1 LASER PRODUCT

Velodyne LiDAR, Inc. 345 Digital Drive, Morgan Hill, CA 95037 / lidar@velodyne.com / 408.465.2800 | www.velodynelidar.com


ANYbotics