POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

Low cost Human Motion Tracking System via Infrared(IR) sensor array

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Supervisor:
Prof. Andrea Bonarini
Student:
Yuting Wang, matricola 822471

Anno Accademico 2016-2017

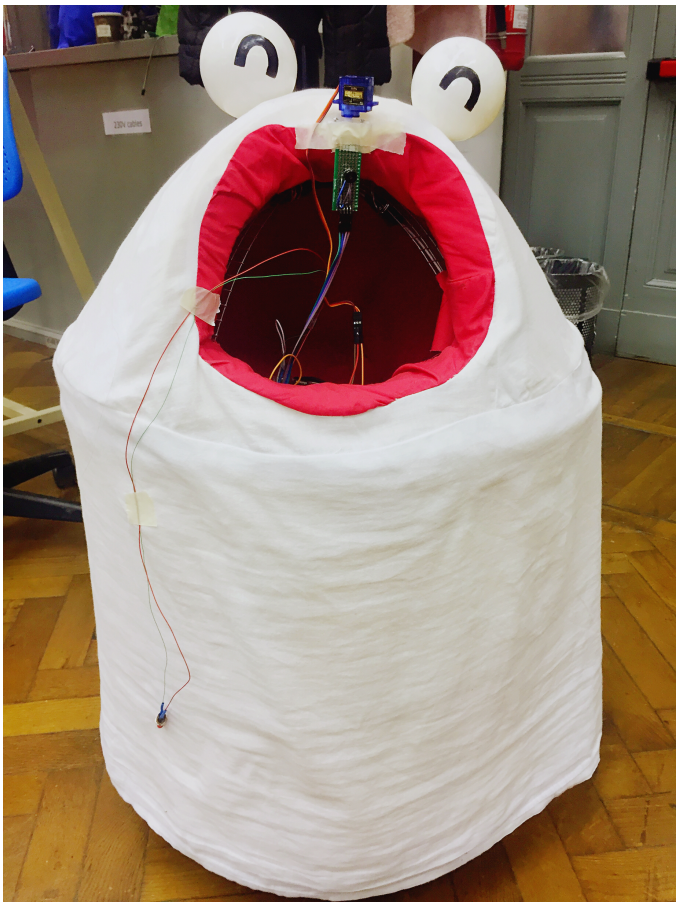


Figure 1: The Robot used in this project

Abstract

Robots are used more and more to take a relevant part in people's life. Together with the explosion and improving of daily life services and manufacturing demands, human-following robots are used in numerous applications. In recent times, some robots are found their place also in therapeutic processes.

This project is aimed at the development of an innovative sensor system to support a robot in following a person. To achieve this, a people detector using an infrared sensor array has been implemented. The result of this detector is sent to a tracking algorithm using thresholding segmentation and a proportional controller to control the robot. This system can make the robot follow a person that walks in front of it.

Keywords: People detection; People tracking; Infrared sensor array; Human-following robot

Sommario

I robot sono sempre più usati per avere una parte rilevante nella vita delle persone. Con l'esplosione e il continuo miglioramento dei servizi per la vita di tutti i giorni e le richieste nelle applicazioni industriali, robot in grado di seguire le persone sono usati in molte applicazioni. Recentemente, sono stati impiegati anche in applicazioni terapeutiche.

Questo progetto aveva lo scopo di sviluppare un sistema sensoriale innovativo per supportare il robot nell'inseguimento di una persona. Per ottenere questo, è stato implementato un rilevatore di persone basato su una matrice di sensori infrarossi. Il risultato di questo sistema viene mandato a un algoritmo di inseguimento che usa una segmentazione a soglia e un controllore proporzionale per controllare il robot nell'inseguimento. Questo sistema permette al robot di inseguire una persona che cammina di fronte a sé.

Parole chiave: Rilevamento di persone; Tracciamento di persone; Matrice di sensori infrarossi; Robot per l'inseguimento di persone

Acknowledgement

First of all, I would like to thank my supervisor, Professor Andrea Bonarini, for his guidance throughout my research, for his help to make this project possible and for his valuable feedbacks during the project. Professor Bonarini has always been patient and encouraging with great advice.

Thanks also go to my friends, lab mates and the department faculty and staff for making my time at Politecnico di Milano be a great experience.

Finally, thanks to my mother and father for their encouragement patience and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For a list of all the ways
technology has failed to
improve the quality of life,
please press three.

Alice Kahn

Robotics is today a quickly evolving world. There are more and more tasks in the world which are done by robots. Robots are used in many fields to assist humans in different tasks. During the last decade, the robotics community interest in social robotics has grown greatly. [3]. Specifically, rehabilitation robotics constitutes an emerging area of research, where the aim is to include robotics technology in the time-consuming and labour-intensive therapeutic process. Research find that robotic therapy may complement other treatment approaches by reducing motor impairment in persons with moderate to severe chronic impairments [5]. The robot in this project is aiming to accompany children with neurodevelopmental disorders, including, and mainly focusing on, children with motor impairments, during the therapy.

1.1 Problem statement

This thesis work deals with the issues related to developing a low-cost and relatively reliable "person following" behaviour that allows the robot to accompany a human. Once the target person is detected, the robot attempts to drive directly towards the person's location and keep a certain distance with the person.

Most of the systems for detecting and tracking people on a mobile robot use either range sensors such as laser scanners [6] and [10], or a colour camera as the primary sensor.

Approaches based on laser scans usually try to detect the legs of a person. However, leg detection in a two-dimensional scan at a fixed height does not provide very robust features for discriminating between people and other moving or fixed objects, so that false positive classifications may occur.

A visual sensor provides richer information, and methods for people detection in colour images that extract features such as skin colour, motion and depth information have been used on mobile robots. Many of these approaches assume that people are close to the robot and face toward it so that methods based on skin colour and face detection can be applied. All methods for detecting and tracking people in colour images on a moving platform face similar problems, and their performance depends heavily on the current light conditions, viewing angle, distance to persons, and variability of appearance of people in the image.

Thermal vision overcomes some of the problems related to colour vision sensors, since humans have a distinctive thermal profile compared to non-living objects, and the sensor data does not depend on light conditions. The primary requirement of this research has been to investigate the realization of a human tracking system based on low-cost sensing devices. Therefore, we chose low-cost IR sensor array instead of thermal vision camera. This type of sensors is finding a great number of applications in human-robot interaction, among others that of gesture recognition in cars. Their application in service and assistive robotics also brings the advantage of exploiting the possibility of some analysis of body configuration, without challenging privacy issues, since no specific person can be recognized through the analysis of this type of images.

1.2 Assumptions and delimitations

Our system only works in the following conditions:

- The robot only moves on a planar ground and in an indoor environment.

- The temperature of the environment is strictly below 30 degrees Celsius. The best environment temperature should be stable around 25 degrees Celsius.

- The target person wears light summer clothes with exposed arms and neck to be better distinguished from the environment.

1.3 Outline

The thesis consists of seven chapters.

In chapter 2 related background knowledge is described. Firstly, a brief introduction to infrared sensor array, Raspberry Pi and Robot Operative System (ROS) is given. Secondly, we provide an overview of people detection and tracking algorithms.

Chapter 3 presents the components of the sensor platform and explains the calculating process of temperature seen by the IR sensor.

Chapter 4 introduces and compares some common robot platforms and discusses the kinematic theory of the robot platform we used in this project.

In Chapter 5, a theoretical analysis and design of the proposed solution is undertaken.

Experimental results are presented in Chapter 6, together with an analysis of the system limitations.

Chapter 7 provides conclusions and recommendations for future approaches, based on the findings of this study.

Chapter 2

Background Knowledge

I do think, in time, people will have,
sort of, relationships with certain kinds of robots
- not every robot, but certain kinds of robots
- where they might feel that it is a sort of friendship,
but it's going to be of a robot-human kind.

Cynthia Breazeal

2.1 Infrared (IR) sensor array

A sensor array is a group of sensors, usually deployed in a certain geometry pattern, used for collecting and processing electromagnetic or acoustic signals. Infrared (IR) sensor array, namely, consists of a number of thermopile sensor elements included in a sensor chip.

Infrared sensor: An infrared sensor is an electronic instrument which is used to sense certain characteristics of its surroundings by either emitting and/or detecting infrared radiation. Infrared sensors are also capable of measuring the heat being emitted by an object and detecting motion.

Infrared Radiation Theory: Infrared waves are not visible to the human eye. In the electromagnetic spectrum, infrared radiation can be found between the visible and microwave regions. The infrared waves typically have wavelengths between 0.75 and 1000μm.

The wavelength region which ranges from 0.75 to 3μm is known as the near infrared regions. The region between 3 and 6μm is known as the mid-infrared and infrared radiation which has a wavelength greater higher than 6μm is known as far infrared.

Infrared technology finds applications in many everyday products. For instance, television sets use an infrared detector to get the signals sent from a remote control. The key benefits of infrared sensors include their low power requirements, their simple circuitry and their portable features.

The Working Principle of Infrared Sensors: All objects which have a temperature greater than absolute zero (0 Kelvin) possess thermal energy and are sources of infrared radiation as a result. Infrared sensors typically use infrared lasers and LEDs with specific infrared wavelengths as sources. Optical components are used to converge or focus the infrared radiation. In order to limit spectral response, band-pass filters can be used. Next, infrared detectors are used in order to detect the radiation which has been focused.

Types of Infrared Sensors: Infrared (IR) sensors fall into two main categories, thermal and photon:

Photon: In this class of detectors the radiation is absorbed within the material by interaction with electrons. The observed electrical output signal results from the changed electronic energy distribution. The photon detectors show a selective wavelength dependence of the response per unit incident radiation power. They exhibit both perfect signal-to-noise performance and a very fast response. But to achieve this, the photon detectors require cryogenic cooling. Cooling requirements are the main obstacle to the more widespread use of IR systems based on semiconductor photo detectors making them bulky, heavy, expensive and inconvenient to use.

Thermal: Thermal infrared detectors are distinguished by the advantages of a wide wavelength response, no requirement for cooling, high-temperature stability, high signal-to-noise ratio and low cost. Thermal sensors are differentiated between pyroelectric and thermopiles:

- **Pyroelectric Sensors:** Here the heat radiation collected by the pyroelectrical material generates a static voltage signal across the crystalline material. Under constant illumination, however, the signal declines, and this makes a periodical refresh necessary. Pyroelectric detectors are applicable for mass production. They have slowly found yet their way into the consumer market through applications in burglar alarm systems and automatic light switches. Pyroelectric detectors are used in high performance gas analyzers, flame detection devices and scientific instrumentation. On the other hand, for static temperature measurements one still needs a relatively expensive setup which

includes mechanical parts.

- **Thermopile Sensors:** The Seebeck effect describes the electric current in a closed circuit composed of two dissimilar materials when their junctions are maintained at different temperatures. This phenomenon is applied extensively to temperature measurement by wire thermocouples. The thermopiles or thermocolumn comprises a series of thermoelements, each element being a thin wire made of two materials of different thermal activity. When a temperature difference occurs between the two ends of a wire, an electrical tension (thermotension) develops. The hot junctions are concentrated on a very thin common absorbing area, while the cold junctions are located on a surrounding heat sink with high thermal mass.

Modern semiconductor technology makes it possible to produce thermopile sensors consisting of hundreds of thermocouples on an area of several square millimetres. Such a sensor is extremely sensitive, shows a very fast response time and due to its smallness, and it is additionally inexpensive because of the employment of semiconductor mass production means and of photolithography.

Many applications are being developed, such as thermography, human detection, night vision, and so on. Quantification of energy allows users to determine the temperature and thermal behaviour of objects. Infrared thermal sensing and imaging instruments make it possible to measure and map surface temperature and thermal distribution passively and non-intrusively.

MLX90621 Family: The traditional thermopile sensor is provided with a shielding cap with filters infrared radiation for transmission, blocking the rest. For instance, visible light may be blocked by a silicon filter, which lets substantially solely IR radiation pass. The shielding cap is typically thermally isolated from the sensing element, for instance by leaving a spatial gap of air or vacuum between the cap and the sensing element. The IR radiation is then collected by the sensing element, which heats up due to the collected energy. The heating results in a temperature change. This temperature change is detected by the hot contact of a thermocouple. The difference of temperature between the sensing element (hot contact) and a reference (a cold contact in the case of thermocouples) produces a readable electrical voltage signal. Hence the measurement value of the voltage is a value representative for the temperature of the heat source.

The hot contact is attached to a membrane which absorbs the infrared radiation and the cold contact is attached to the bulk matrix of the de-

vice. Several factors contribute to errors in the measurement. The sensing element may receive parasitic signals from sources other than the object of interest. The increase of the reference temperature may also affect the measurement. In case of thermocouples, the bulk reference temperature may increase (for instance, due to environmental conditions) and create an offset, thus diminishing the accuracy of the measurement.

To reduce the offset the MLX90621 family comprises a first substrate, an active primary sensing element and a cap with a window allowing IR radiation to reach the active sensing element. The sensing element comprises a thermocouple with its could junction attached to the first reference substrate. An additional passive secondary detector is allowed to receive IR radiation from the first reference substrate, so the offset is compensated by substracting the signal of the additional detector from the signal of the active sensing element. [[8]] A further discussion about offset compensation during temperature calculation is presented in Chapter 3.

2.2 Target detection and tracking

Target detection and tracking are the most fundamental tasks in computer vision. Target detection is the process of detecting the instances of a certain class of objects (e.g., humans, cars) in signal prints, typically images and video. Target tracking is the process of locating moving objects in different frames of a video while maintaining the correct identities.

Target detection and tracking are usually coupled together to locate the objects of interest through the video. We need to first detect the target in each video frame, and then track them across different frames. In this dissertation, we focus on the object class of human.

2.2.1 Object segmentation

Object segmentation is also an important task in computer vision. Object Segmentation is the process of delineating the target object from the image. Human segmentation can benefit many computer vision applications, such as motion analysis, body tracking, and person recognition. In this dissertation, we also adopted the basic theory of object segmentation, which will be described in detail in a later chapter.

Segmentation techniques are either contextual or non-contextual. The latter takes no account of spatial relationships between features in an image,

and group pixels together on the basis of some global attribute, e.g. grey level or colour. Contextual techniques additionally exploit these relationships, e.g. group together pixels with similar grey levels and close spatial locations.

We choose the easiest non-contextual thresholding in the project. With a single threshold, it transforms a greyscale or colour image into a binary image considered as a binary region map. The binary map contains two possibly disjoint regions, one of them containing pixels with input data values smaller than a threshold and another relating to the input values that are at or above the threshold (figure 2.1). The former and latter regions are usually labelled with zero (0) and non-zero (1) labels, respectively. The segmentation depends on image property being thresholded and on how the threshold is chosen.

(a) Original (b) Greyscale (c) Segmentation

Figure 2.1: Example of Image Segmentation By using Thresholding

2.2.2 Movement detection using background subtraction

Background subtraction is a widely used approach for detecting moving objects in videos from static cameras. The rationale in the approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called The "background image", or "background model" [9]. Movement detection is a lot more difficult with a moving camera, as all the scene is moving. The background image, which is a representation of the scene with no moving objects, thus have to be kept regularly updated so as to adapt to the varying luminance conditions and geometry settings.

2.2.3 Movement detection using optical flow

Optic flow is defined as the change of structured light in the image, e.g., on the retina or the camera's sensor, due to a relative motion between the eyeball or camera and the scene. Figure 2.2a shows the shift of two visual features (star and hexagon) on a plane and their angular displacements on the surface of the eyeball. Figure 2.2b shows three frames, which show the movement of the silhouette of a head. The optic flow is depicted as the correspondence of contour pixels between frame 1 and 2 as well as frame 2 and 3. For methods estimating flow, the challenge is to find the point correspondence for each pixel in the image, not only the contour pixels.

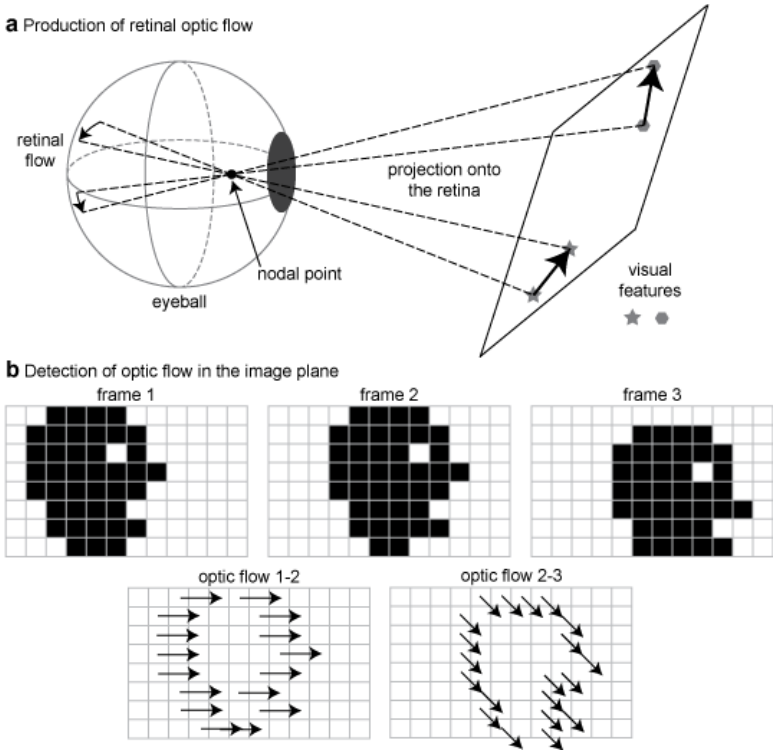


Figure 2.2: The production and detection of optic flow

The optical flow describes the direction and time rate of pixels in a time sequence of two consequent images. A two dimensional velocity vector, carrying information on the direction and the velocity of motion is assigned to each pixel in a given place of the picture. By estimating optical flow between video frames, we can measure the velocities of objects in the video.

The method based on optical flow is complex, quantitative use of the data will also require quantitative predictions of accuracy, thus resulting in

high memory requirements, in-turn resulting in high cost.

2.2.4 Thermal image based human detection

Any object whose temperature is above absolute zero Kelvin emits radiation at a particular rate and with a distribution of wavelengths. This wavelength distribution is dependent on the temperature of the object and its spectral emissivity. The spectral emissivity, which can also be considered as the radiation efficiency at a given wavelength, is in turn characterized by the radiation emission efficiency based on whether the body is a black body, grey body, or a selective radiator. Around room temperature, the typical emissions for a solid matter are maximal in the long wave infrared region of the electromagnetic spectrum (7μm to 14μm).

Only limited visual information can be captured by CCD cameras under poor lighting and weather conditions. Meanwhile, the brightness intensities of thermal images are representatives of the temperatures of object surface points. Pedestrians typically emit more heat than background objects, such as trees, road, etc. Image regions containing pedestrians or other "hot" objects will be brighter than the background. Hence theoretically, infrared thermal images can be a reliable source for human detection at night-time and in bad weather conditions. Thermal images, compared to visible images, lack several features, such as color and texture information, which plays a vital role for human detection and classification. The most discriminative and distinctive features of human beings from the background lie in their contours.

2.2.5 Object tracking

Tracking can be defined as the problem of estimation of the trajectory of an object in the image plane as it moves around a scene. In other words, a tracker assigns consistent labels to the tracked objects in different frames of a video. Additionally, depending on the tracking domain, a tracker can also provide object-centric information, such as orientation, area, or shape of an object. Tracking objects can be a complex activity due to:

- loss of information caused by projection of the 3D world on a 2D image,
- noise in images,
- complex object motion,

Figure 2.3: The Raspberry Pi 3 board

- scene illumination changes, and
- real-time processing requirements.

In this project, we try the best to overcome these complexities. The usage of IR sensor solved the illumination problem. We consider the child to be tracked is always very close by, and that is the closest person to the robot, thus the noise in the thermal image is negligible compared with the target. The algorithm is based on the simple proportional control to reduce the calculation.

2.3 Raspberry Pi

Raspberry Pi is a small, powerful, cheap, hackable and education-oriented computer board introduced in 2012. It operates in the same way as a standard PC, requiring a keyboard for command entry, a display unit and a power supply. This credit card-sized affordable computer with good performance is perfect platform for interfacing with many devices. The vast majority of the system's components - its central and graphics processing units, audio and communications hardware along with 512 MB memory chip, are built onto single component. The Raspberry Pi board shown in Figure **??** and in Figure 2.4 contains essential (processor, graphics chip, program memory - RAM) and other optional devices (various interfaces and connectors for peripherals). The processor of Raspberry Pi is a 32 bit, 700 MHz System on a chip, which is build on the ARM11 architecture. SD Flash memory serves as a hard drive to Raspberry Pi's processor. The unit is powered via the micro USB connector while Internet connectivity may be via an Ethernet/LAN cable or via a USB dongle.

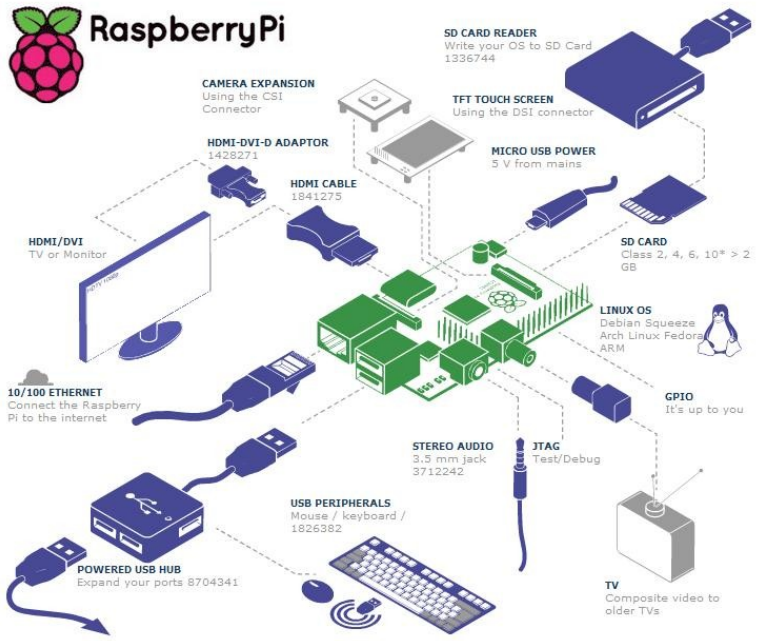


Figure 2.4: The Raspberry Pi's core-components

The Raspberry Pi, like any other computer, uses an operating system. The official operating system for Raspberry Pi is a version of Linux Debian, followed by a list of operating systems also supported. With the release of the Raspberry Pi 3 and its ARM Cortex-A53 based BCM2837 processor, it is now possible to run Ubuntu directly on the Raspberry Pi.

Ubuntu Linux is chosen as the operating system in this thesis project since it is the officially claimed as the operating system supporting ROS, the system we selected to implement robotic functionalities.

Python is the preferred language for Raspberry Pi. In order to better support C programming, we also needed to install a C library called bcm2835 library. It provides access to GPIO and other IO functions on the Broadcom BCM 2835 chip, as used in the RaspberryPi, allowing access to the GPIO pins on the 26 pin IDE plug on the RPi board so that it is possible to control and interface with various external devices.

It provides functions for reading digital inputs and setting digital outputs, using SPI and I2C, which is the most important usage in the project.

Another important library is OpenCV, which is an image and video processing library with bindings in C++, C, Python and Java. OpenCV is used for all sorts of image and video analysis, like facial recognition and detection, photo editing, advanced robotic vision, and a whole lot more. The

python-OpenCV library is used in this thesis project for image processing.

2.4 Robot Operative System(ROS)

2.4.1 Overview

The Robot Operating System was first developed on 2008, by a group of engineers and scientists from Willow Garage as a new kind of Operative system fully oriented to robotics, that also tries to follow the trends and updates in hardware and software day to day (Willow Garage, 2013). ROS is completely open source (BSD) and free to use, change, and commercialize upon. ROS was chosen as the main Operative system to be used during this thesis project.

As ROS is organized on Stacks and packages, during this thesis work rosserial is the mainly used package. Rosserial is a protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or network socket. In this project, rosserial sets the communication between the detection system and the tracking motion.

2.4.2 Nodes

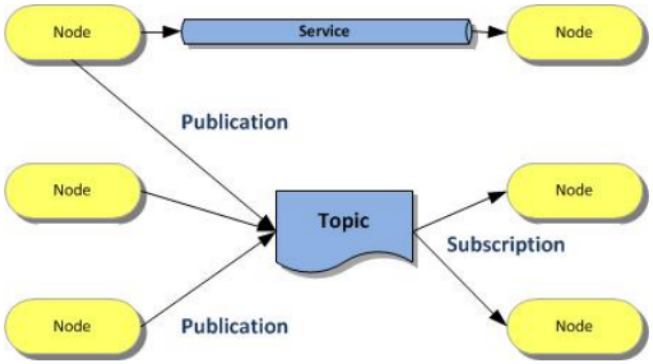


Figure 2.5: Illustration of communication using topics or services

A ROS node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise

many nodes. To put it simply, A node is an executable that uses ROS to communicate with other nodes.

We use two nodes in this thesis work, a publisher and a subscriber. A publisher ("talker") node broadcasts messages to a topic while a subscriber ("listener") receives messages from a topic.

Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data, subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to the same topic.

A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Msg files are simple text files for specifying the data structure of a message.

Chapter 3

Sensor Platform

You could claim that moving from pixelated perception, where the robot looks at sensor data, to understanding and predicting the environment is a Holy Grail of artificial intelligence.

Sebsatian Thrun

3.1 Selection of IR Sensor

Sensor selection means meeting requirements. The list of requirements should be exhaustive, involve the user, working environment and costing. To meet the requirements we need to spend some time just to sort and identify the different IR sensors in the market and find one that would be the best for our application (detecting and tracking an object).

In this section, we are going to compare 4 different IR sensors: Omron D6TL-06, Omron D6T-8L-06, Panasonic AMG8832 and Melexis MLX90620. Table3.1 displays some important characteristics of these sensors.

Model	**pixel**	**Field of View**	**Accuracy**	**Frame Rate**
D6T-44L-06	4×4	44.2° × 45.7°	±1.5°C	4FPS
D6T-8L-06	1×8	62.8° × 6.0°	±1.5°C	4FPS
AMG8832	8×8	60° × 60°	±3.0°C	1 to 10FPS
MLX90621	16×4	120° × 30°	±1.0°C	0.5 to 512FPS

Table 3.1: Some important characteristics of different IR sensor

MLX90621 has the best frame rate among all the 4 sensors. However, we do not need a very high frame rate, 4 FPS compared with the human

(a) Omron D6T (b) AMG88

Figure 3.1: Example of Omron D6T series thermal sensor and Panasonic Infrared array sensor Grid-EYE (AMG88)

walk speed is an acceptable frame rate. The pixel numbers directly affects the resolution: the more pixels we have, the better resolution we obtain. AMG8832 and MLX90621 both have the maximum pixels (64 pixels). We will discuss in the next section the reason why we choose MLX90621, mostly for its Field Of View (FOV).

There are some other factors not mentioned in the table: temperature measurement range, supply voltage and costing. A brief explanation of them is given below:

- **Temperature measurement range:** each sensor has a wide temperature measurement range, covers the expected room temperature and human body temperature.
- **Supply voltage:**Raspberry Pi could provide either 5.0 V or 3.3 V supply voltage. Thus all the sensors could meet the requirement directly or with a simple circuit.
- **Costing:**all the 4 sensors are low-cost sensor. There is no much difference between the prices.

3.2 Melexis MLX90621

The IR sensor used in this project is Melexis MLX90621. The MLX90621 has improved considerably on speed and temperature resolution (x4) compared to the previous generation products. This exciting high speed product updates and further broadens the application potential of low cost thermal imaging. It contains 64 IR pixels with dedicated low noise chopper stabilized amplifier and fast ADC integrated. A PTAT (Proportional To Absolute Temperature) sensor is integrated to measure the ambient temperature of the chip. The outputs of both IR and PTAT sensors are stored in internal RAM and are accessible through I^2C (Figure 3.2)

Parameter	Details
Supply Voltage	2.6V
Number of thermopiles	16 × 4
NETD	0.20K RMS @4Hz refresh rate
Field Of View	120° × 25°, 60° × 16°, 40° × 10°
Programmable Frame Rate	0.5Hz...512Hz
Operating Temperature Range	-40...+85°C
Storage Temperature Range	-40...+125°C

Table 3.2: Melexis MLX90621 specification

The device specifications are shown in Table3.2.

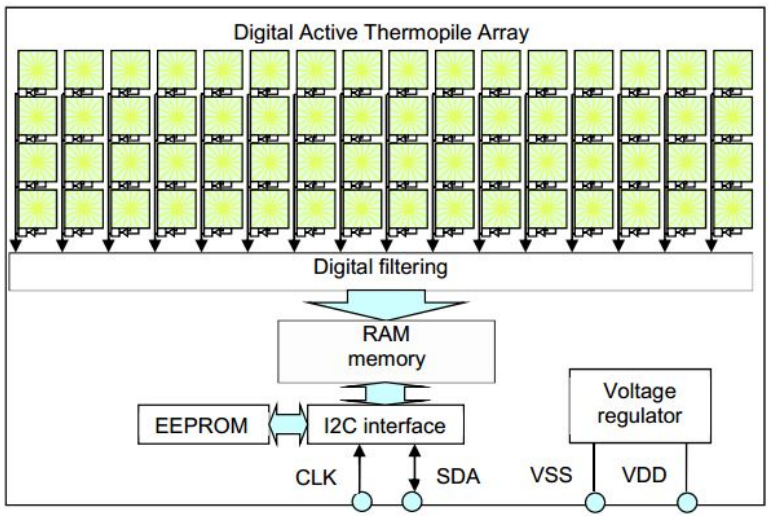


Figure 3.2: Structure of MLX90621

3.2.1 Field of View

Field of View (FOV) is defined as the solid angle through which a detector is sensitive to radiation. (Figure 3.3)

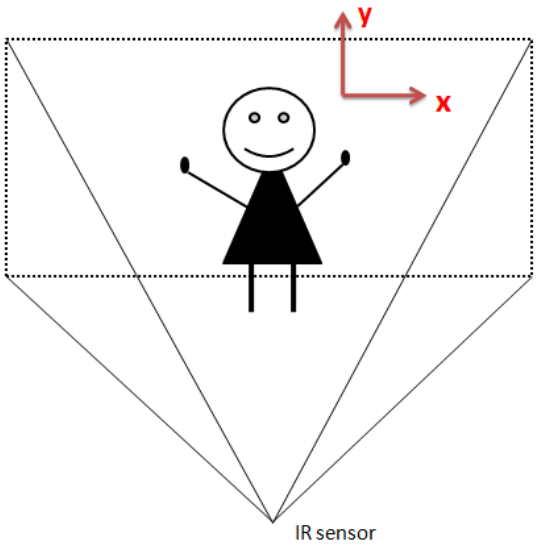


Figure 3.3: Field Of View Diagram

As shown in Table 3.2, there are three options of Field of View(FOV). Due to our application conditions, the height of the robot is fixed and also the variation of height of the target (since they are humans) is negligible compared to the sphere of movement. As long as we considered only the horizontal movement in our project, the height of the target became a less important parameter. Thus, the horizontal direction (x direction) is much more important than the vertical direction (y direction). Considering also the sensitivity of the specified FOV, as shown in Figure 3.4, in order that more pixels could have a better sensitivity in the X direction, $120° \times 25°$ is the best choice for our application.

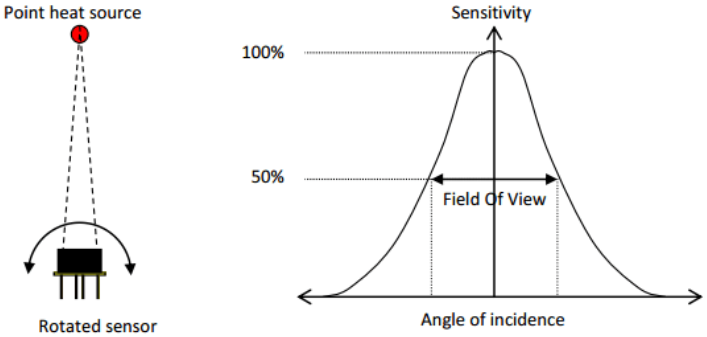


Figure 3.4: Field of View measurement

3.3 Hardware setup

3.3.1 Pin definition and description of MLX90621

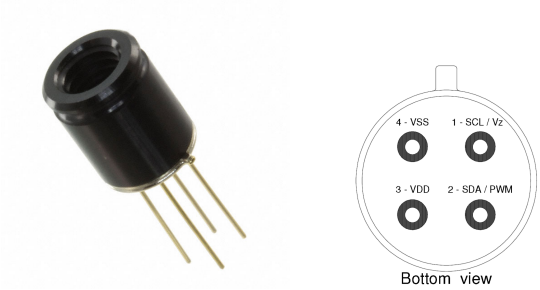


Figure 3.5: PIN description of MLX90621

Pin Name	Function
SCL	Serial clock input for 2 wire commnications protocol
SDA	Digital input / output 2 wire communications protocol
VDD	External supply voltage
VSS	Ground

Table 3.3: PIN description of MLX90621

3.3.2 GPIO layout of Raspberry Pi

The Raspberry Pi offers up its GPIO over a standard male header on the board, along the top edge. We are using Raspberry Pi3 in this thesis project, which contains 40 pins.(Figure 3.2)

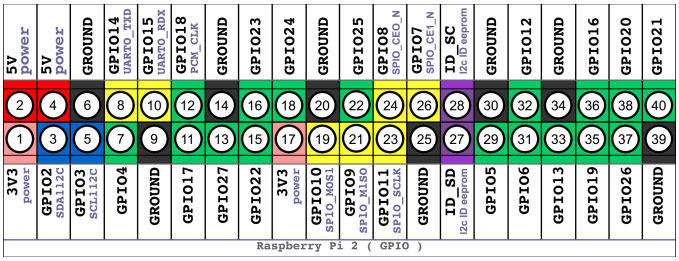


Figure 3.6: Raspberry Pi3 GPIO Header

These pins are a physical interface between the Pi and the outside world. Of the 40 pins, 26 are GPIO pins and the others are power or ground pins(plus two ID EEPROM pins)

When programming the GPIO pins there are two different ways to refer to them: GPIO numbering and physical numbering. GPIO numbering are the GPIO pins as the computer sees them. The numbers don't make any sense to humans, they jump about all over the place, so there is no easy way to remember them. The other way to refer to the pins is by simply counting across and down from pin 1 at the top left(nearest to the SD card). This is physical numbering and it looks like as shown in Figure 3.6.

3.3.3 Hardware Setup

The IR sensor node consists of the following hardware parts.

- Raspberry Pi3
- Melexis MLX90621 IR array
- Rectifier 1N4001

Figure 3.3 is the circuit diagram of the IR sensor board. The IR sensor array MLX90621 is connected to the Raspberry pi through I2C. I2C is a two wire bus, the connections are called SDA (Serial Data) and SCL (Serial Clock). Each I2C bus has one or more masters (the Raspberry Pi here) and one or more slave devices(the MLX90621 here). With I2C, every device has an address that each communication must be prefaced with. The IR sensor array defaults to an address of 50, 51, 52, 53, 54, 55, 56, 57 and 60.

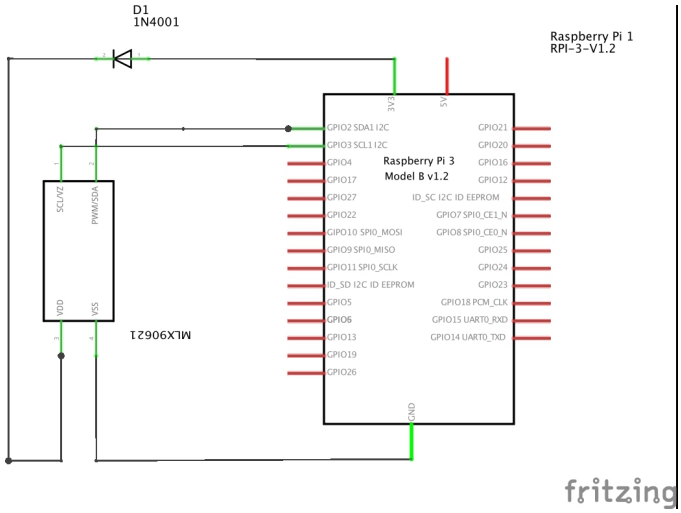


Figure 3.7: Circuit diagram of IR sensor node

As the preferred operating voltage of the sensor is 2.6V, it's prudent to connect a small rectifier diode in series with the positive power offered by Raspberry Pi(3.3V) (a 1n4001 works nicely.)

After enabled the I2C drivers on the Raspberry Pi, we could now detect the chip with the command:

sudo i2cdetect -y 1

Once the command is executed, a grid of dashes with numbers where any i2c device is detected would be shown in the terminal. (Figure 3.8)

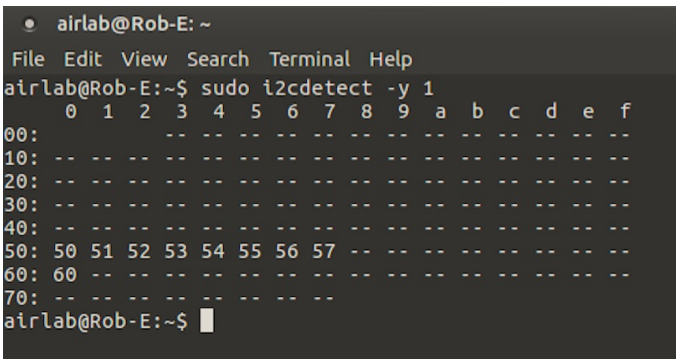


Figure 3.8: I2C addresses of MLX90621 detected by RPi

3.4 Principle of Operation and Calculation

3.4.1 Principle of operation

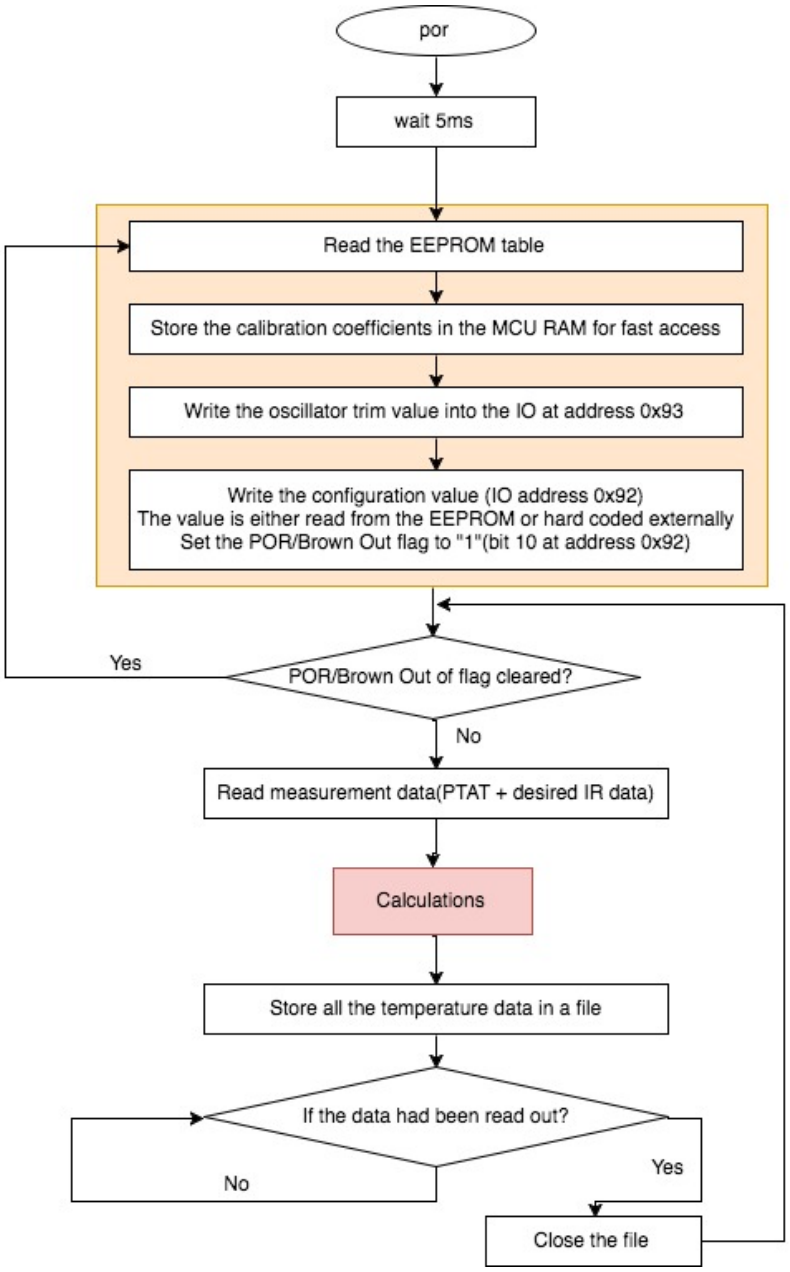


Figure 3.9: Operation block diagram

The output of all IR sensors and absolute temperature sensors is scanned according to the programmed refresh rate. Using their output data as well as calibration constants written in EEPROM the absolute chip temperature and object temperature, 'seen' by each pixel can be calculated. For this goal several sequential calculations must be done according to Figure 3.9 Operation block diagram.

The Power On Reset(POR at the beginning) is connected to the VDD supply. The device will start approximately 5ms after the POR release.

After the POR is released the external MCU must execute an initialization process. For the sake of brevity, the part of initialization and reading measurement data (RAM data) will not be covered here. In the next section, the part of temperature calculation will be introduced in detail.

3.4.2 Temperature Calculation

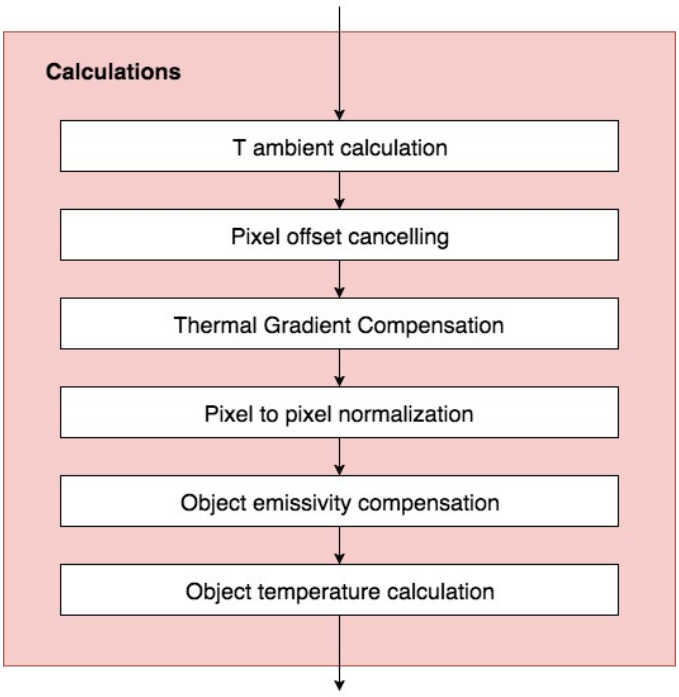


Figure 3.10: Calculation part block diagram

The array consists of 64 IR sensors (also called pixels). Each pixel is identified with its row and column position as $P_{ix(i,j)}$ where i is its row number (from 0 to 3) and j is its column number (from 0 to 15).

ConfigRegister	$PTAT\_data$	$V_{TH}(25)$	K_{T1}	K_{T2}
00	0x0CFB	3323.750	10.69189	0.00144
01	0x19F7	6647.500	21.38379	0.00288
10	0x33EF	13295.000	42.76758	0.00577
11	0x67DE	26590.000	85.53516	0.01153

Table 3.4: Constants' values at different resolution settings

The output signal of the IR sensors is relative to the cold junction temperature. That is why we need first to calculate the absolute chip temperature Ta(sensor temperature), by using the following formular:

$$T_a = \frac{-K_{T1} + \sqrt[2]{K_{T1}^2 - 4K_{T2}[V_{TH}(25) - PTAT\_data]}}{2K_{T2}} + 25, [^\circ C]$$

Constants $V_{TH}(25)$, K_{T1}, K_{T2} and $PTAT\_data$ are related to the maximum resolution set in the configuration register (ConfigRegister), as shown in table3.4

Afterwards, the temperature seen by each pixel can be calculated by using the formula:

$$T_{O(i,j)} = \sqrt[4]{\frac{V_{IR(i.j)COMPENSATED}}{\alpha_{comp(i,j)} * (1 - K_{s^4}) + S_{x(i.j)}} + T_{ak^4}} - 273.15, [^\circ C]$$

Where:

$V_{IR(i,j)COMPENSATED}$ is the parasitic free IR compensated signal
$\alpha_{comp(i,j)}$ is the compensated sensitivity coefficient for each pixel
K_{s^4} is the compensation factor for the sensitivity
$T_{ak^4} = (T_a + 273.15)^4$ where T_a is the ambient temperature
$S_{x(i,j)} = K_{s^4} * \sqrt[4]{\alpha_{comp(i,j)}{}^3 * V_{IR(i,j)COMPENSATED} + \alpha_{comp(i,j)}{}^4 * T_{ak^4}}$

$V_{IR(i,j)COMPENSATED}$ is a combination of 3 different kinds of compensations, which are offset compensation, thermal gradient compensation(TGC) and emissivity compensation.

Offset compensation: The nonuniformity (NU) noise in IR sensor array, which is due to pixel-to-pixel variation in the detectors' response, can considerably degrade the quality of IR images since it results in a fixed-pattern noise (FPN) that is superimposed on the true image.

One blind infrared radiation detector having a radiation responsive element is provided and shielded from infrared radiation of the scene. The

blind infrared radiation detector provides a proportional electrical signal in response to infrared radiation incident thereto [11]. The blind infrared radiation detector is configured to provide a signal indicating thermal distortion of the infrared radiation detector array, thus to reduce fixed pattern noise. The compensation formula is shown below:

$$V_{IR(i,j)OffsetCompensated} = V_{IR(i,j)} - (A_{i(i,j)} + B_{i(i,j)} * (T_a - T_{a0})) \quad (3.1)$$

Where:

$V_{IR(i,j)}$ is an individual pixel IR_data readout (RAM read)

$A_{i(ij)}$ is an individual pixel offset restored from the EEPROM using the formula: $A_{i(i,j)} = \frac{A_{common}+\delta A_{i(i,j)}*2^{\delta A_{i_{scale}}}}{2_{3-ConfigReg[5:4]}}$. A_{common} is the minimum offset value, $\delta A_{i(i,j)}$ is the difference between the individual offset and the minimum value, $\delta A_{i_{scale}}$ is the scaling coefficent. All these three parameters are stored in the EEPREOM.

$B_{i()i,j}$ is an individual pixel offset slope coefficent.

T_a is the ambient temperature calculated before

$T_{a0} = 25°C$ is a constant

Thermal gradient compensation (TGC): used to compensate the errors due to "thermoelectric effects".

Thermoelectric Effects: errors occur when more than one type of metal is used in the construction of the sensing element and associated body and wiring. When more than one metal is used, the different metals can produce a Seebeck voltage that would lead to a temperature gradient along the sensing element and associated wires.

$$V_{IR(i,j)_TGCCompensated} = V_{IR(i,j)\_OffsetCompensated} - TCG * V_{IRcp_OffsetConpensated} \quad (3.2)$$

Where:

$V_{IRcp_OffsetConpensated}$ is the offset compensated IR signal of the thermal gradient compensation pixel.

$TGC = \frac{TGC_{EEPROM}}{32}$, TGC_{EEPROM} is a coefficient stored at EEPROM.

Emissivity compensation According to the Stefan-Boltzmann Law, the emissive power of a blackbody is $E_b = \sigma T^4$, where

$$\sigma = 5.10670 \times 10^{-8} \frac{W}{m^2} K^4$$

and the temperature is measured in Kelvin. Accordingly, for a non-blackbody, the emissive power is $E_b = \varepsilon\sigma T^4$, where ε is the emissivity of the surface being considered.

If emissivity is not taken into account, the surface temperature of an object may appear to be emitting more radiation than it really is, due to the addition of reflective radiation for instance. This can give an incorrect perception of the emissive power or temperature of that surface. This can be especially true for an IR sensor that has been calibrated to a nearly standard blackbody surface. The simple correction that is commonly made, dividing the measured temperature by the target emissivity, is shown blow:

$$V_{IR(i,j)_{COMPENSATED}} = \frac{V_{IR(i,j)_{TGCCompensated}}}{\varepsilon} \tag{3.3}$$

For the sake of simplicity, all the unmentioned parameters default to scale coefficients or scaled values stored into EEPROM as either two's complement value or unsigned value, thus we will not further introduce their calculation formulas in detail.

3.4.3 Temperature Storage

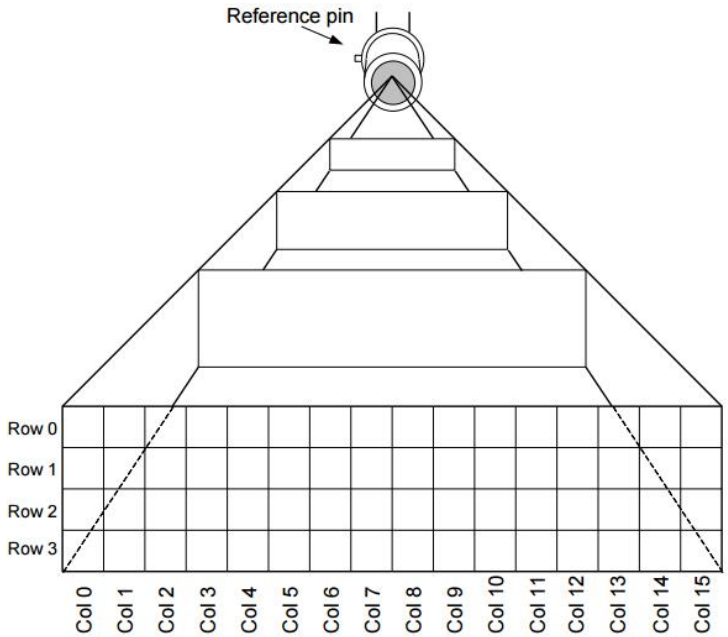


Figure 3.11: Pixel position in the whole FOV

We created an unsigned array and stored all 64 calculated temperature into it. Note, here we stored the temperature value as Kelvin hundredths of degree.

```
unsigned short temperaturesInt[64];
temperaturesInt[x]=(unsigned short)((to+273.15)*100.0);
```

These data would be used in the image process for human detection and tracking. In order to protect the integrity of the 64 values (one frame), we make a FIFO file each time we have a new frame and unlink it at the end of this frame. A "FIFO" is a special file type that permits independent processes to communicate. One process opens the FIFO file for writing, and another for reading, after which data can flow as with the usual anonymous pipe in shells or elsewhere. Unlink a file means that, if no processes have that file open, it is deleted.

```
char mlxFifo[] = "/var/run/mlx90621.sock";
char mlxFifo[] = "/var/run/mlx90621.sock";
mkfifo(mlxFifo, 0666);
if((fd=open(mlxFifo, $O_WRONLY|O_CREAT$))==-1)
    {printf("Open Error");}
write(fd,temperaturesInt,sizeof(temperaturesInt));
close(fd);
unlink(mlxFifo);
```

Chapter 4

Wheeled Robot Platform

There are three types of land-based robots, wheeled robots, tracked robots and leg-robots. The wheeled ones are by far the most popular mobile robots due to the following advantages:

- Usually low-cost compared to other method
- Simple design and construction
- Abundance of choice

4.1 Types of Wheeled Robots

Wheeled robots can use any number of wheels to navigate, with minimum of one to maximum of whatever the number we choose to fit specifications.

Single wheel robots are highly unstable and require extreme engineering and design techniques. Two wheeled robots might sound simpler than one wheeled robot but are still harder to balance and keep upright. Usually, the center of gravity of these robots is kept above the wheel axis and the wheels are attached parallel to each other.

Three wheeled robots are easier to build compared to the previous two. These robots do not require any specialized balancing algorithms and this is why they are the preferred choice for a beginner. They can either be differentially steered or can use a tricycle approach. The wheels are normally arranged in a triangular manner and are hence balanced.

Four wheels are more efficient compared to three or two wheels. The first two of the four wheels can be used to steer and the next two to drive the robot. Balancing a four wheeled robot is never an issue.

The main disadvantage compared to a three wheeled robot is the extra cost of the fourth wheel and an extra motor to drive it.

Another disadvantage is the need to keep the 4 wheels all on the ground, which requires suspensions.

More wheels are not really required since this would add additional cost for wheels, motors, additional power, additional computation and complex design. The only exception concerns the movement of heavy weights.

4.2 Types of Robot Wheels

There are different kinds of wheels to choose from, for a Wheeled Mobile Robot (WMR).

Standard wheel has two degree of freedom and can traverse Front or Reverse. The center of the wheel is fixed to the robot chassis. The angle between the robot chassis and wheel plane is constant.

Figure 4.1: Standard wheel

Orientable wheels are mounted to a fork which holds the wheel in place. Orientable wheels are normally used to balance a robot and rarely used to drive a robot in a configuration called synchro-drive.

Figure 4.2: Ball wheel

Ball wheels contain a spherical metal or nylon ball (or any hard spherical material these days) positioned within a holder. The ball has 360° of freedom and is normally used to balance a robot. The disadvantage is that these wheels may have high friction and require power to push. They are also not suitable for uneven, dusty, and greasy surfaces. Ball wheels are also generally referred to as "castor ball wheels".

Figure 4.3: Omni wheel

The best choice for a robot that requires multi-directional movement. Omni wheels are normal wheels with passive wheels (rollers) attached around the circumference of the center wheel. Omni wheels can move in any direction and exhibits low resistance when they move in any direction. The small wheels are attached in such a

way that the axis of the small wheels are perpendicular to the axis of the bigger center wheel which makes the wheel to rotate even parallel to its own axis.

4.3 Wheel Control Theory

In the previous section we have seen the different types of wheels and their arrangements. Since omni wheels have smaller wheels attached perpendicular to the circumference of another bigger wheel, they allow wheels to move in any direction instantly. The major advantage is that they do not need to move in any direction without changing the orientation. We chose omni wheeled platform in this thesis project.

4.3.1 Omni-wheel Design: 3 Wheels vs. 4 Wheels

Generally Omni wheeled robots use either a three wheeled platform or a four wheeled platform.

3-wheel design: A three wheel design offers greater traction as any reactive force is distributed through only three points and the robot is well balanced even on uneven terrain. This design also reduces an additional wheel compared to a 4 wheeled robot which makes it cost effective (omni wheels are expensive).

4-wheel design: In 4 wheel design, 4 Omni wheels are attached at 90° to each other. This means any two wheels are parallel to each other and other two wheels perpendicular. The first and the major benefit is the simplified calculation. Since there are two pairs of wheels, each pair requires only one calculation and all four wheels require only two calculations.

But a four-wheeled Omni robot does not balance on irregular terrain and also not all four wheels are guaranteed to stay on the same plane. Additional wheel also calls for an extra cost. Since Omni wheels are a combination of many wheels / rollers into one, there is a greater resistance to rotation which leads to greater loss of energy compared with three-wheeled Omni robot, when not driving parallel to a pair of wheels.

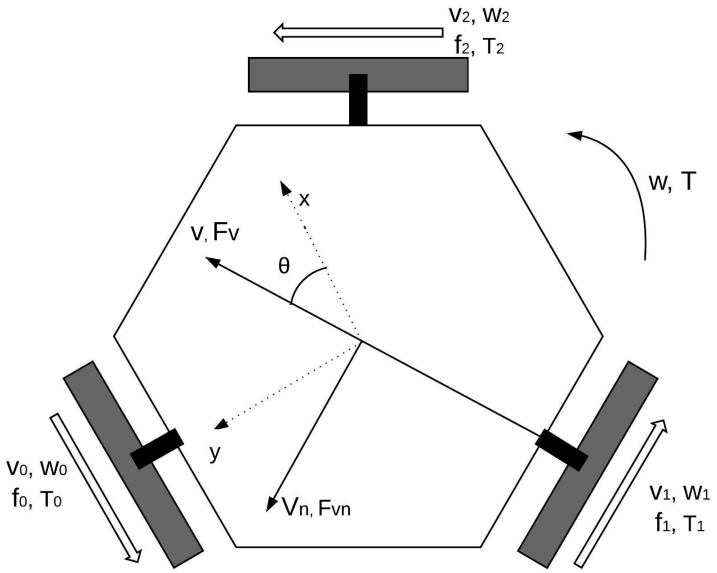


Figure 4.4: Configuration of three wheeled robot

We decided to use 3-wheeled Omni robot platform during our project because of the low-cost and simpler mechanics. Figure4.4 present the configuration of the three wheeled robot, as well as all axis and relevant forces and velocities of the robotic system, detailed as follows:

- x, y, θ - Robot's position (x,y) and θ angle to the defined front of robot;
- $d[m]$ - Distance between wheels and center robot;
- $v_0, v_1, v_2[m/s]$ - Wheels linear velocity;
- $\omega_0, \omega_1, \omega_2[rad/s]$ - Wheels angular velocity;
- $f_0, f_1, f_2[N]$ - Wheels traction force;
- $T_0, T_1, T_2[Nm]$ - Wheels traction torque;
- $V, V_n[m/s]$ - Robot linear velocity;
- $\omega[rad/s]$ - Robot angular velocity;
- $F_V, F_{V_n}[N]$ - Robot traction force along V and V_n;
- $T[Nm]$ - Robot torque (respects to ω).

4.3.2 Holonomic and Non-Holonomic Drive

Non-Holonomic Drive: If the controllable degree of freedom is less than the total degrees of freedom, then it is known as non-holonomic drive. A car has three degrees of freedom; i.e. its position in two axes and its orientation. However, there are only two controllable degrees of freedom which are acceleration (or braking) and turning angly of steering wheel. This makes it difficult for the driver to turn the car in any direction (unless the car skids or slides).

Holonomic Drive: Holonomic refers to the relationship between controllable and total degrees of freedom of a robot. If the controllable degree of freedom is equal to total degrees of freedom, them the robot is said to be Holonomic. A robot built on omni-wheels which is used in this project is a good example of holonomic drives as it can freely move in any direction and the controllable degrees of freedom is equal to total degrees of freedom.

4.3.3 Kinematic Model

In order to find motion models for a surface vehicle, the pose of the vehicle must be identified as (x, y, θ) and associated velocities are $v_x(t) = \frac{dx(t)}{dt}, v_y(t) = \frac{dy(t)}{dt}, \omega(t) = \frac{d\theta(t)}{dt}$. The following test uses the notation presented in Figure4.4, where the defined "front" also defines the v direction and its orthogonal vn direction.

Equation (4.1) allows the transformation from linear velocities v_x and v_y on the static(world) axis to linear velocities V and V_n on the robot's axis.

$$\begin{bmatrix} V(t) \\ V_n(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} cos(\theta(t)) & sin(\theta(t)) & 0 \\ -sin(\theta(t)) & cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_x(t) \\ v_y(t) \\ \omega(t) \end{bmatrix} \tag{4.1}$$

WWheels' velocities v_0, v_1 and v_2 are related with robot's velocities V, V_n and ω as described by Equation(4.2).

$$\begin{bmatrix} v_0(t) \\ v_1(t) \\ v_2(t) \end{bmatrix} = \begin{bmatrix} -sin(\pi/3) & cos(\pi/3) & d \\ 0 & -1 & d \\ sin(\pi/3) & cos(\pi/3) & d \end{bmatrix} \cdot \begin{bmatrix} V(t) \\ V_n(t) \\ \omega(t) \end{bmatrix} \tag{4.2}$$

Applying the inverse kinematics is possible to obtain the equations that determine the robot's velocities related with the wheels velocities. Solving for V, V_n and ω, the following can be found:

$$V(t) = (\frac{\sqrt{3}}{3}) \cdot (v_2(t) - v_0(t)) \tag{4.3}$$

$$V_n(t) = (\frac{1}{3}) \cdot (v_2(t) + v_0(t)) - (\frac{2}{3}) \cdot v_1(t) \tag{4.4}$$

$$\omega(t) = (\frac{1}{3 \cdot d}) \cdot (v_0(t) + v_1(t) + v_2(t)) \tag{4.5}$$

The dynamical equations relative to the accelerations can be described in the following relations:

$$M \cdot \frac{dV(t)}{dt} = \sum F_V(t) - F_{BV}(t) - F_{CV}(t) \tag{4.6}$$

$$M \cdot \frac{dV_n(t)}{dt} = \sum F_{V_n}(t) - F_{BV_n}(t) - F_{CV_n}(t) \tag{4.7}$$

$$J \cdot \frac{d\omega(t)}{dt} = \sum T(t) - T_{Bw}(t) - T_{C\omega}(t) \tag{4.8}$$

Where the following parameters relate to the robot are:

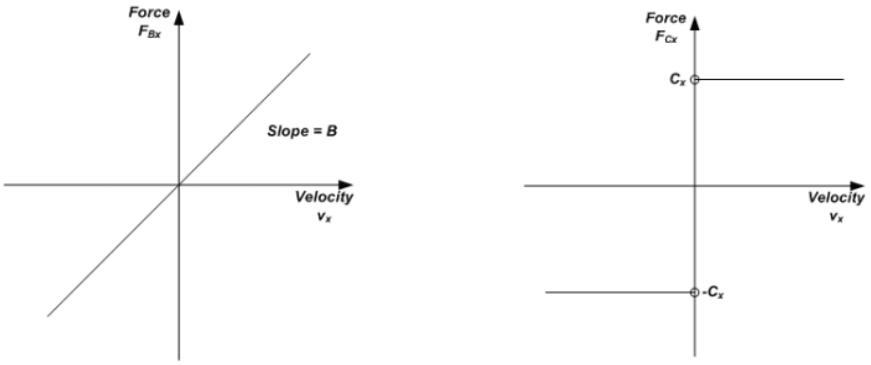
- $M[Kg] - Mass$;
- $J[kgm_2]$ - Inertia moment;
- $F_{BV}, F_{BV_n}[N]$ - Viscous friction forces V and V_n;
- $T_{B\omega}[Nm]$ - Viscous friction torque with respect to the robot's rotation axis;
- $F_{CV}, F_{CV_n}[N]$ - Coulomb frictions forces along V and V_n;
- $T_{C\omega}[Nm]$ - Coulomb friction torque with respect to robot's rotation axis.

Viscous friction forces are proportional to robot's velocities, see Figure4.5(a), and such as:

$$F_{BV}(t) = B_c \cdot V(t) \tag{4.9}$$

$$F_{BV_n}(t) = B_{V_n} \cdot V_n(t) \tag{4.10}$$

$$T_{B\omega}(t) = B_\omega \cdot \omega(t) \tag{4.11}$$



(a) Viscous friction (b) Coulomb friction

Figure 4.5: Frictions

Where the following parameters relate to the robot as follows:

- $B_V, B_{V_n}[N/(m/s)]$ - Viscous friction coefficients for directions V and V_n;
- $B_\omega[Mn/(rad/s)]$ - Viscous friction coefficient to ω

The Coulomb friction forces are constant in amplitude, see Figure4.5(b)

$$F_{CV}(t) = C_V \cdot sign(v(t)) \tag{4.12}$$

$$F_{CV_n}(t) = C_{CV} \cdot sign(V_n(t)) \tag{4.13}$$

$$T_{C\omega}(t) = C_\omega \cdot sign(\omega(t)) \tag{4.14}$$

where the following parameters relate to the robot as follows:

- $C_V, C_{V_n}[N]$ - Coulomb friction coefficient for directions V and V_n;
- $C_\omega[Nm]$ - Coulomb friction coefficient for ω

The relationship between the traction forces and rotation torque of the robot with the traction froces on the wheels is described by the following equations:

$$\sum F_V(t) = (f_2(t) - f_0(t)) \cdot sin(\frac{\pi}{3}) \tag{4.15}$$

$$\sum F_{V_n}(t) = -f_1(t) + (f_2(t) + f_0(t)) \cdot cos(\frac{\pi}{3}) \tag{4.16}$$

$$\sum T(t) = (f_0(t) + f_1(t) + f_2(t)) \cdot d \tag{4.17}$$

The traction force on each wheel is estimated by traction torque, which can be determined using the motor current, as described in the following equations:

$$f_j(t) = \frac{T_j(t)}{r} \tag{4.18}$$

$$T_j(t) = 1 \cdot K_t \cdot i_j(t) \tag{4.19}$$

where:

- 1 - Gearbox reduction;
- r [m] - Wheel radius;
- $K_t[Nm/A]$ - Motor torque constant;
- $i_j[A]$ - Motor current (j=motor number).

4.4 Rapid Robot Prototyping framework (Nova Core)

Nova Core (Rapid Robot Prototyping) is a modular HW/SW system, developed at AIRLab, which enables to assembly the electronics of a robot in few hours [2]. Each hardware module implements a functionality for a robot (e.g., motor control, IMU, sensor data collection, wifi, GPS, ...) and includes an ARM processor that runs a real-time operating system. All modules are connected via CAN bus on which a real-time protocol has been implemented (see Figure 4.6 for reference). Modules can also be integrated in the micro-ROS environment and that can connect them as ROS nodes to a ROS-based system. Figure 4.7 shows one of Nova Core modules, with shared components highlighted.

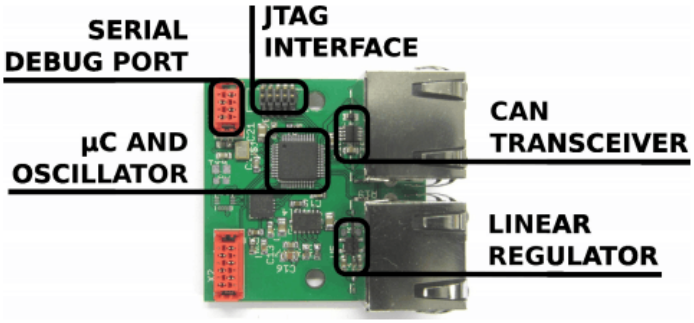


Figure 4.7: A Nova Core hardware module, with components from the reference design highlighted

The power supply module (PS) is responsible of powering the Nova Core bus and all connected modules. It sports a switching DC/DC converter, with a wide range of power inputs and a stable 5 V output, so that it can be used to power up our Raspberry Pi.

The other important Nova Core module is the DC motor controller board (DCM), to satisfy one of the most common primary platform requirements: drive a motor and let the robot move.

The inertial measurement unit module (IMU) allows the measurement of acceleration, angular velocity, and bearing with respect to the heart magnetic field, together with the altitude of a robot.

Proximity module used to interface with proximity sensors. Each module connects to up 4 sensors. Calibration and data filtering algorithms run on

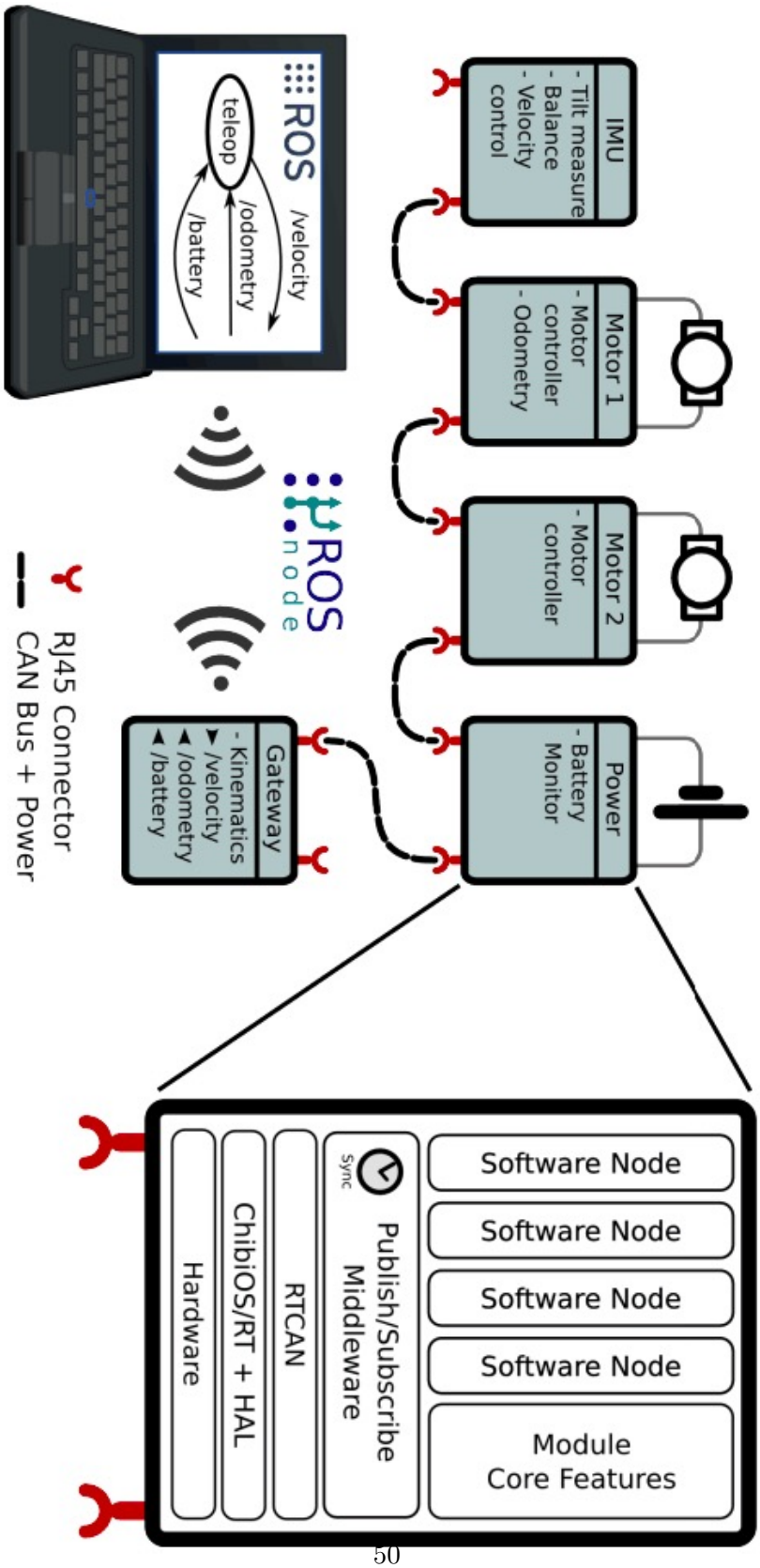
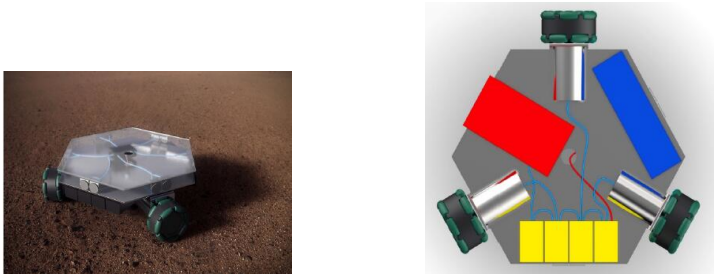


Figure 4.6: Architecture of a balancing robot developed with the Nova Core framework

the microcontroller, which produces distance measurements.

4.5 Hardware Setup



(a) Hardware Setup (b) Hardware Structure

Figure 4.8: Hardware Set-up of the wheel-platform

As shown in Figure4.8, the metal plate has a side of 15cm and hexagonal shape. Wheels of 7cm diameter with connections at L of 4 cm length are fixed at 3 sides of the hexagon. The cylindrical motors occupy about 6.5cm of the base. The 3 zones created by the motors are occupied by:

- A battery (colored blue) with dimensions of 154×27×44mm (Turnigy nano-tech 5000mah 3S 40-80C)
- A block of 3 cards for motors (colored yellow)

The final implementation is shown blow (Figure4.9):

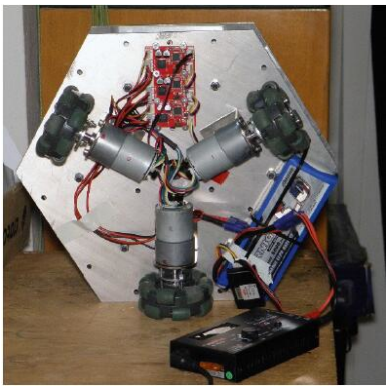


Figure 4.9: Hardware implementation

Chapter 5

Method and Implementation

Despite all of our technological advances, content creation still requires time, inspiration, an a certain amount of sweat. There aren't any shortcuts. You can't write a algorithm for it. You can't predict it. You can't code it.

Shawn Amos

The main part of this project consisted in the design of an efficient algorithm to follow people for a robot with an infrared sensor array. As explained in Chapter 2, an infrared sensor provides temperature data, which are useful to detect the targeted person and estimate movement. By creating a Publisher node, the position information of the target is published to the topic "PositionInfo". By creating a Subscriber node, the position information of the target is subscribed and used in the following algorithm.

5.1 Overall Algorithm

In the proposed method, the robot receives messages from the human detection process. At the first frame a person is in the field of view of the IR sensor array. Then, for the next frames, the tracking algorithm is used to know from frame to frame where the person is in the frame. Either the person is detected in the image or the tracker can lose the person. In this case the tracker goes back to the people detection process.

In this section, the overall algorithm is described. Then, few sections explain more in details the different steps of the algorithm including the special cases encountered and how they can be solved.

The algorithm consists of two parts. After the initialization, first the human detection process finds a target. Then, the main loop runs and tries to track the target. Figure 5.1 illustrates this.

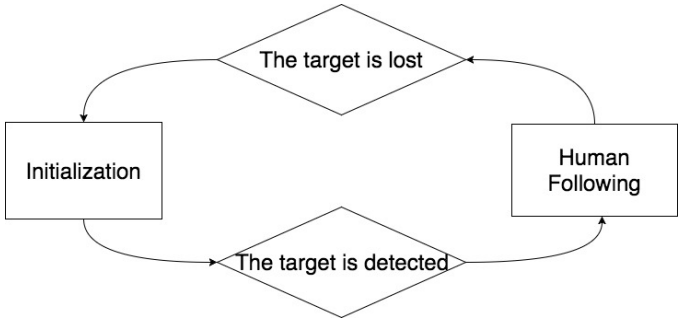


Figure 5.1: Organization of the algorithm

5.2 Initialization

The first part of the algorithm is the initialization step. At this stage the IR array sensor is initialed and all the important flags are set to the initial value, for example:

- **Target_Exist:** flag to estimate if there is a eligible target exist in the field of view. The initial value is set to "None". Once a target is found, this flag is set to "1".
- **Intrudew_Flag and Intrudex_Flag:** flags to estimate if there is a unexpected "Intrude" ("Intrudew" or "Intrudex"). The initial value is set to "None" respectively until a "Intrude" is detected.
- **Intrudew_count and Intrudex_count:** when a "Intrude" occurs, the "Intrude counter" starts to count the frame until the "Intrude" finished or reached the maximum waiting time. The counters are set to "0" as initial value.
- **y_target:** set to None as initial value to estimate if it is the first time we enter the main loop. Once the frame appeals, the "y_target" get a value.
- **Positioninfo.flag:** the flag shows the robot condition. The initial value is set to "0" (Stop state).

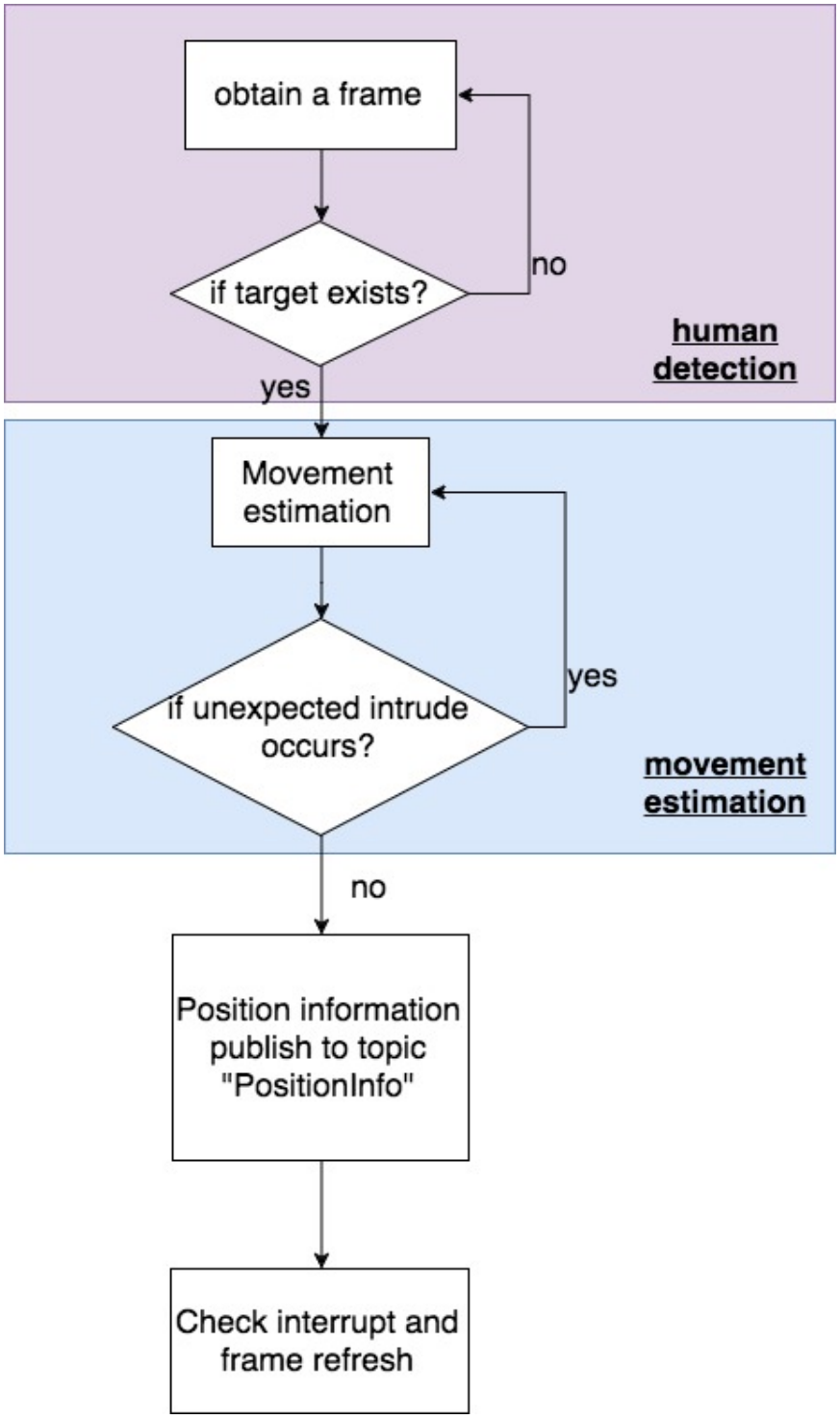


Figure 5.2: Flowchart for the Publisher Node

5.3 Human Detection and Movement Estimation

As shown in Figure5.2, the Publisher node deals with the questions of "where is the target person" and "where does this person go", respectively. These two classes of questions are closely related, but have clearly different objectives.

5.3.1 Human Detection

As first step, we need to get the temperature data from the FIFO file (mlx90621.sock), which we have described in Chapter 3. At the very beginning, it is reasonable that we consider there is no target person. Thus we create a flag "Target_Exist" and set the default value as "None" until we find an eligible target.

At the beginning, we call it "Target searching" state, namely, we need to define a proper method to find an eligible target. There are some researches related to human body tracking using infrared sensor array. [7] developed a method which localizes heat sources using a far-infrared sensor array. [1] simply selects a sensor which outputs the highest temperature to localize a heat source. Thus, it does not consider occlusions nor the effect of other heat sources.

Firstly considering that our robot would closely follow its target and keep the distance within one meter, we simply count the total number of the pixels which see a "higher temperature". Although the temperature of a person is as known 37 degrees centigrade, we still need to pay attention to some objective factors:

- The normal body temperature (37°C) as most people think is oral temperature(by mouth). But the temperature measured by the IR sensor array is the skin surface temperature that is lower than the oral temperature.
- It is clear that clothing also affects the amount of IR radiation emitted, so that the amount and color can make a small difference in sensor performance.
- The room temperature is around 25°C

In summary, we define the "higher temperature" as the temperature above 28 degrees centigrade.

As we have already introduced in Chapter 1, the Melexis MLX90621 is a 16×4 Infrared sensor array. We make following assumptions:

- The robot always tries to keep a distance within 1.0 meter with the target.
- The robot always tries to keep the target in the middle of FOV (Field of View).
- The robot mainly enrolled in the therapy for children around 10 years old. The average shoulder to shoulder width of a 10 years old children is 11.25 inches (28.575 cm)[4]. We use 30 cm for simplicity.
- The diameter of the IR sensor is negligible compared to 30 cm.

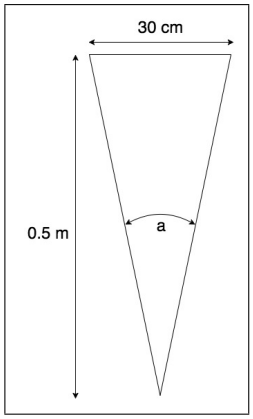


Figure 5.3: Eligible target definition

As shown in Figure 5.3, a is the angle corresponding to our eligible target mapping on the FOV of the sensor (equivalent to a point):

$$tan(\frac{a}{2}) = \frac{15cm}{0.5m}$$
$$a \approx 33.3^\circ$$

In chapter 2, Figure 3.4 shows the diagram of FOV. The total visual angular along x axis (16 columns in total) is 120°, namely, 7.5° per pixel. Thus 33.3° is equivalent to 4 columns (4×4=16 pixels). Considering the possible floating range and the decreasing resolution with distance, we want to set a low limitation during the "Target Searching" state, we use 3 columns (3×4=12 pixels) instead of 4 columns. (Code of process are presented below).

```
fifo=open('/var/run/mlx90621.sock','r')
Target_Exist=None
while Target_Exist is None:
    ir_raw=fifo.read()
    ir_trimmed=ir_raw[0:128]
    ir=np.frombuffer(ir_trimmed,np.uint16)
    T30=0
    for pixel in ir:
        if pixel>30315:
```

```
        T30=T30+1
    if T30<12:
        Target_Exist=None
    else:
        Target_Exist=1
```

5.3.2 Movement Estimation

Once we have located our target, the next step is to estimate the movement of the target. In order to do this, we first have to arrange the 64 temperature numbers we obtained from the FIFO file to a "16×4" array (one frame).

```
ir=ir.reshape((16,4))[::-1,::-1]
ir=img_as_float(ir)
p2,p98=np.percentile(ir,(2,98))
ir=exposure.rescale_intensity(ir,in_range=(p2,p98))
```

We compute the 2^nd and 98^th percentiles of array elements respectively and then normalize the intensity levels to the range from the 2^nd percentile (p2) to the 98^th percentile (p98). After this process, the array becomes a set of numbers from 0 to 1. So that we can easily map it to a grey-scale image array.

Image segmentation: As we have mentioned in chapter 2, we choose the simplest non-contextual thresholding in the project. With a single threshold, it transforms our greyscale image array into a binary image considered as a binary region map. The binary map contains two possibly disjoint regions, one of them containing pixels with input data values smaller than a threshold and another relating to the input values that are at or above the threshold. The former and latter regions are usually labeled with zero (0) and non-zero (1) labels, respectively. The threshold is chosen depending on the high temperature we defined before for the human body.

Contour Tracing: Also known as border following or boundary following; contour tracing is a technique that is applied to digital images in order to extract their boundary. The boundary of a given pattern P, is the set of border pixels of P. The contour is the only thing we know about the feature of our target in this project and to be used in the movement estimation. To better understand, an example is shown in Figure 5.4.

Figure 5.4: An example shows contour changing as the target person moving

Under our assumed condition, there is only one child with the therapist in the therapy room. In other words, there are maximum two persons in the room that can be detected by the robot. As we have no other information to do human identification, the default target is the biggest contour, i.e the one who is closer to the robot.

OpenCV offers the function boundingRect(), which calculates the minimal up-right bounding rectangle as the contour for the specified point set. There are two useful features we could extract from the contour: 1. the top-left coordinate of the rectangle; 2.its width. These two simple features are two important dependents for our movement estimation. The top-left coordinate expresses the horizontal movement. The width expresses the change of distance.

Intrude: We set 2 flags (Intrudew_flag and Intrudex_flag) to check if overlapping occurs. "Intrude" is defined as a big change either of the width of the contour (target_w) or the top-left coordinate of the rectangle (target_x). This double check uses to avoid the confusion caused by the passed therapist. **Confusion** means that the robot can not distinguish between the target and the intruder. An example of an Intrude is given below. The therapist enters the field of view from left and passes through the target person. In Figure5.5 the therapist first enters the vision field of the robot, but it is negligible compared to the target.

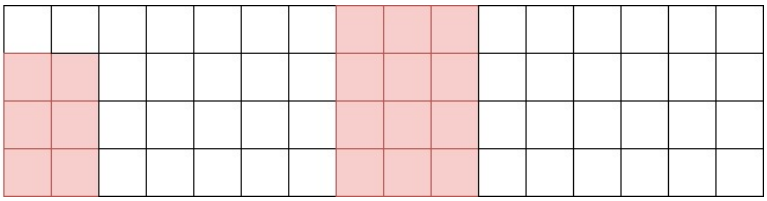


Figure 5.5: The therapist enters the vision field

As shown in Figure5.6, the therapist approaches to the target, and becomes comparable with the target, but not big enough to cause a confusion.

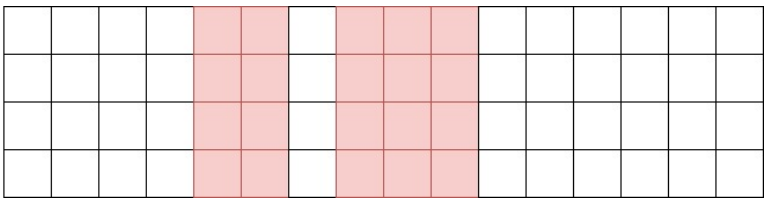


Figure 5.6: The therapist approaches the target person

In Figure5.7, as the therapist gets closer to the target and the robot, a confusion is created due to the emergence of another eligible target. As we defined before, it is an "Intrude_x".

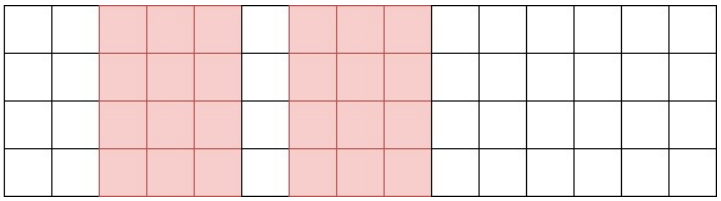


Figure 5.7: An Intrude is generated and created confusion

Figure5.8 shows the overlap condition of the therapist and the target person. Thus an "Intrude_w" is created.

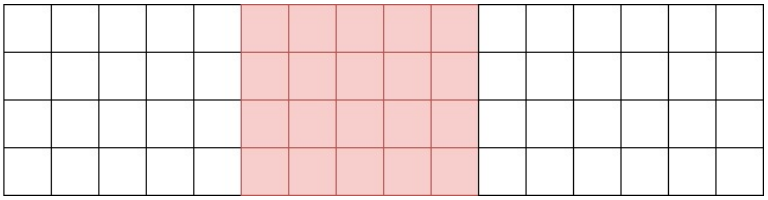


Figure 5.8: An Intrude is generated and created confusion

In Figure5.9, the therapist passed by the target person and the Intrude is vanished.

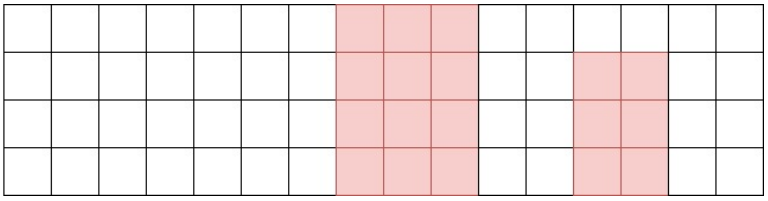


Figure 5.9: An Intrude is generated and created confusion

We make an assumption that the average walking speed for a normal adult is about 1 meter per second and for a 10 years old child is around 0.8 meter per second. 3 second is enough for the therapy to walk out of the vision field of the robot or make a distinguished distance between the child and the therapy. Once we found there is a "Intrude", we set the corresponding flag "Intrude_w" or "Intrude_x" 1 and keep the former position data. We waited for 2 second to see if the "Intrude" disappear. We use two Frame Counters (Intrudew_count and Intrudex_count) to estimate the maximum waiting time. The Frame Rate is 0.08s, thus we wait for maximum 40 new frames. After 40 new frames if the "Intrude" still presents, we assume the robot lost the target and goes back to the "Target Searching" state.

5.3.3 Position Information Publish

After all the calculations and processes above, we obtain the position information of our target. The position information contains 3 values: the middle coordinates of the contour, the width of the contour and the state flag. Our Publish Node (Talker) then send a message included these values to the topic "PositionInfo".

State Flag: PositionInfo.flag Contains the state of the tracking process. When we are in the "Target Searching" state, we set PositionInfo.flag to 2; If there is a "Intrude" Exist, we set the flag to 1; Or we are following the target, the flag is set to 0.

Figure5.10 shows a detailed flowchart of the Talker with it's important Initialized values.

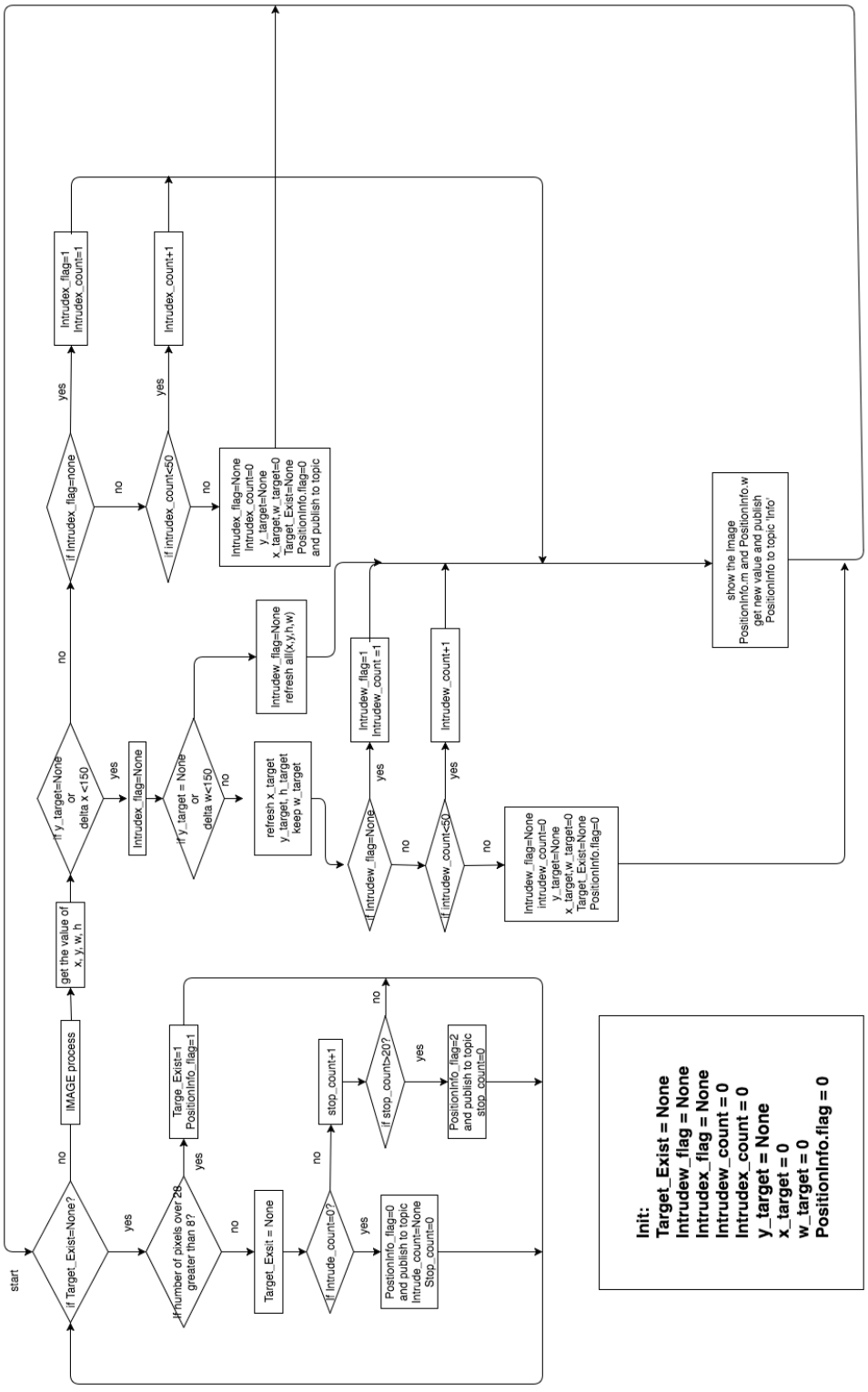


Figure 5.10: Flowchart of the Publisher Node

5.4 Target Tracking and Following

In the previous sections, two parts of the project have been presented. The first one is to process the temperature data and convert to a thermal image. The second one is to detect the target person in the image and estimate movement. This section explains how to track the person's position in the image and how to move the mobile base to follow the target.

5.4.1 Thermal Image Based Servoing

In this section, we use feedback information extracted from IR sensor array to control a DC servo motor.

A servo motor is a combination of DC motor, position control system and gears. Servos have many applications in the modern world and with that, they are available in different shapes and sizes. We will be using SG90 Servo Motor (Figure 5.11) in this section, it is one of the popular and cheapest one. SG90 is a 180 degree servo. So with this servo we can position the axis from 0-180 degrees.

Figure 5.11: TowerPro SG90

A servo motor mainly has three wires, one is for positive voltage, another is for ground and last one is for position setting. The red wire is connected to power, brown wire is connected to ground and yellow wire (or white) is connected to signal, as shown in Figure 5.12.

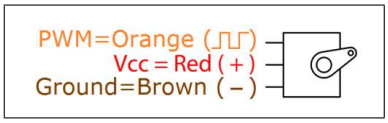


Figure 5.12: Pin diagram of SG90

The frequency of PWM (Pulse Width Modulated) signal can vary based on type of servo motor. For SG90 the frequency of PWM signal is 50Hz (Figure 5.13). Once the frequency is selected, the other important thing here is the DUTY RATIO of the PWM signal.

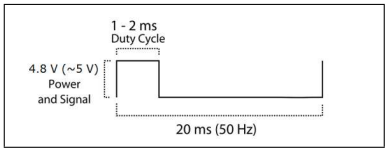


Figure 5.13: PWM period of SG90

The position of the servo motor is set by the length of a pulse. The servo expects to receive a pulse at least every 20 milliseconds. If that pulse is high for 0.8 millisecond, the servo angle will be at position 0 (0°); if it is 1.5 milliseconds, it will be at its center position (90°); and if it is 2.2 milliseconds, it will be at position 2 (180°). The table below shows the Servo

Position for that particular Duty Ratio. We can get any angle in between by choosing the value accordingly.

POSITION	DUTY RATIO
0°	4
30°	4.5
90°	7.5
150°	10.5
180°	11

Table 5.1: Duty Ratio of SG90 servo motor

There are two pins on the Pi that is capable of producing pulses in this way (GPIO pin 12 and pin 18). We choose pin 12 to be connected to the control pin of the servo. The power to the servo is provided by pin 1 of Pi, which provided 5V. Servos require 4.8-6V DC power to the motor, but the signal level (pulse output) can be 3.3V, which is how its OK to just connect the signal line directly to the GPIO output of the Pi. Figure 5.14 shows the pin connection of Raspberry pi and servo motor. The 1kΩ resistor is not essential, but it does protect the GPIO pin from unexpectedly high currents in the control signal, which could occur if a fault developed on the servo.

The codes below shows the initialization of pin 12:

```
servoPIN18 = 12
GPIO.setmode(GPIO.BOARD)
GPIO.setup(servoPIN18, GPIO.OUT)
p18= GPIO.PWM(servoPIN18, 50)
```

The function *GPIO.setmode* set up GPIO numbering schemes.As the main GPIO header of the Raspberry Pi shown in Chapter 3, there are 40 pins in total. The top left pin (as we look at this photo) is called pin 1, the one to the right of it is pin 2. So the next row is 3, 4 etc. and on down to 25, 26. This is how pin headers are numbered. The slightly confusing part is that each pin also has a name, according to what it does. In RPi.GPIO we can use either pin numbers (BOARD) or the Broadcom GPIO numbers (BCM).

```
angle = float(90.0)
p18.start(setAngle(angle))
```

The codes above set the start point at 90°, the center position. As we know, *p18.start(Duty Cycle)* function used to start PWM signal generation. We define a function *setAngle()* to convert angle to Duty Ratio.

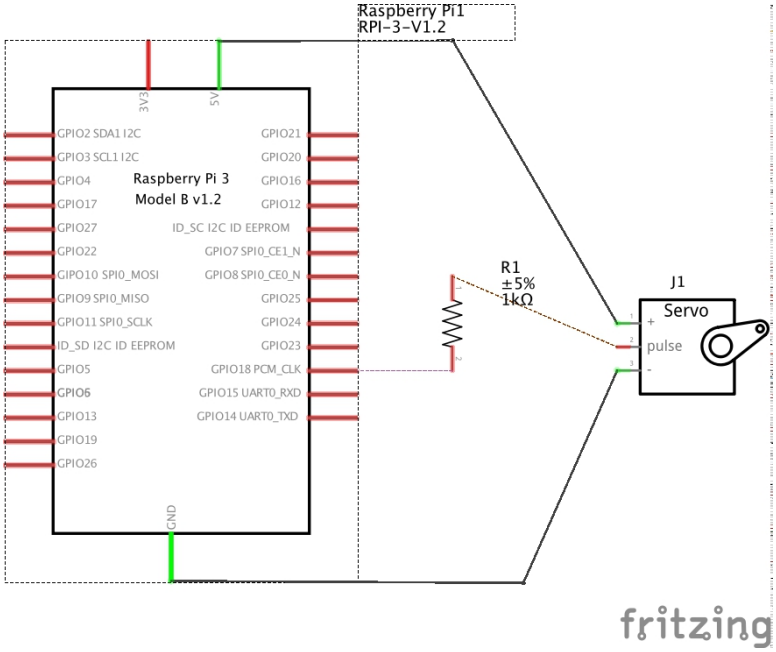


Figure 5.14: servo motor connection

```
def setAngle(angle):
    mode = float(7.5)
    if (angle >= 0) and (angle <= 180):
        mode = 4.5 + (angle-30.0)/20.0
    else:
        mode = 7.5
    return mode
```

At the "Target Searching" state, we added the following code to search the target. When there is no eligible target in the FOV, the motor starts to rotate and tries to find the target.

```
if T30 < 8:
    Target_Exist = None
    if angle >= 10:
        angle = angle-10
    else:
        angle = angle +170
    p18.ChangeDutyCycle(setAngle(angle))
else:
    Target_Exist = 1
```

Once there is an eligible target enters the FOV, the motor starts to follow

the target and tries to keep the target in the middle of the FOV. The code is shown below:

```
if middle > 550.0:
    if angle < 170:
        angle = angle + 10
    else:
        angle = 180
    p18.ChangeDutyCycle(setAngle(angle))
    Target_Exist = None
if middle < 450.0:
    if angle > 10:
        angle = angle - 10
    else:
        angle = 0
    p18.ChangeDutyCycle(setAngle(angle))
    Target_Exist = None
else:
    continue
```

5.4.2 Comparison With Background Subtraction

A common approach to identifying the moving objects is background subtraction, where each frame is compared against a reference or background model. Pixels in the current frame that deviate significantly from the background are considered to be moving objects. In this section, we will make a comparison with background subtraction and explain the reason why we did not choose it.

Background modeling is at the heart of any background subtraction algorithm. Much research has been devoted to developing a background model that is robust against environmental changes in the background, but sensitive enough to identify all moving objects of interest. par In our project, the only reference is the 64 temperature values we captured for each frame. As shown in the previous section, we transfer the temperature values into a image array.

For a mobile robot, the background is always changing, one can hardly have a precise background model and make a background subtraction. Considering the unicity of the dependent feature, it is reasonable to simplify the process, and make background modeling possible. We choose the background reference as the pixels below the threshold temperature.

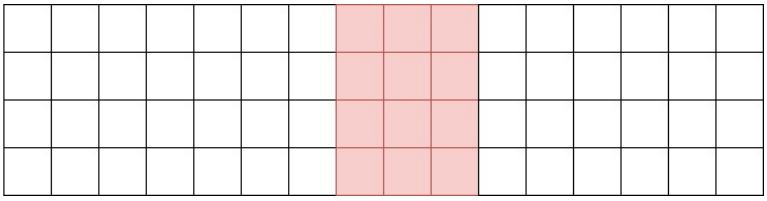


Figure 5.15: First-frame background reference

First-frame subtraction: the first frame is considered as the frame when an eligible target is first detected. For example, Figure 5.15 below is the background reference, the red pixels are the detected target. Our target is always distinguished clearly with respect to the background in the assumed operating environment.

We classify the movement into translation-movement and distance-movement. Translation-movement is defined as the target move to left and right while keeping the same distance from the robot. The distance-movement is defined as the target moves further or closer to the robot without translation.

Figure 5.16 and Figure 5.17 show a leftward translation-movement:

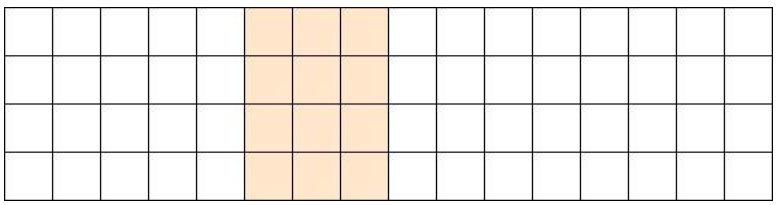


Figure 5.16: A small leftward translation-movement

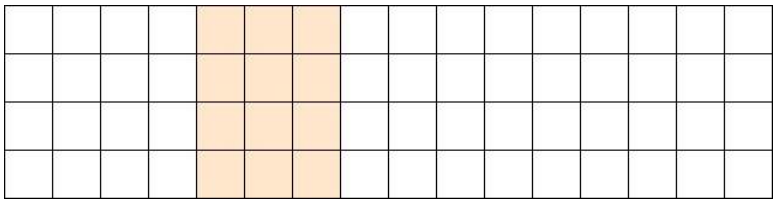


Figure 5.17: A big leftward translation-movement

Figure 5.18 and Figure 5.19 show a distance-movement:

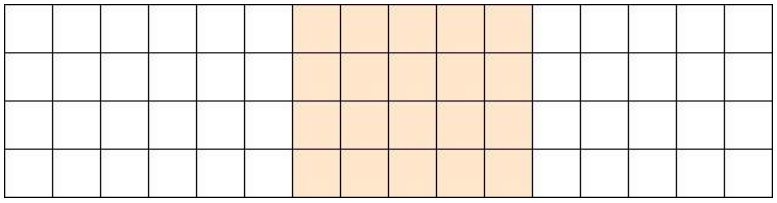


Figure 5.18: A distance-movement moves closer to the robot

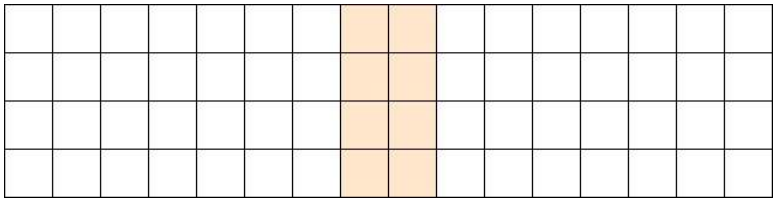


Figure 5.19: A distance-movement moves further to the robot

In the reality, the movement of the target is always the combination of the distance-movement and the translation-movement. But the decoupling of the two movements makes the control algorithm easier. We could also assume the linear velocity and the angle velocity of the robot as two separately controlled uncorrelated variables.

5.4.3 Control Theory

Proportional controller amplifies the error as show in the block diagram below (Figure 5.20):

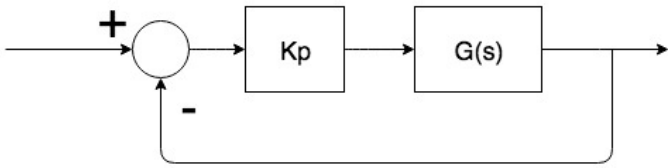


Figure 5.20: Proportional control diagram

So the actuating signal is proportional to the error. In a proportional controller, steady state error (SSE) tends to depend inversely upon the proportional gain, so, if the gain is larger, the error goes down. In this system, SSE is given by the expression:

$$SSE = \frac{1}{1 + KpG(0)}$$

As the proportional gain, Kp, grows, the SSE becomes smaller. As we increase the proportional gain, it provides smaller amplitude and phase margin, faster dynamics satisfying wider frequency band and larger sensitivity to the noise. Namely, larger proportional gain results produce larger changes in response to the error, and thus affects the speed at which the controller can respond to changes in the system. However, despite the reduction, proportional control can never manage to eliminate the steady state error of the system.

First order plants do not have natural oscillations. Proportional control is easy to implement for these systems, and we could set a large Kp to improve the system performance without worrying. However, the situation is different for higher order plants. An example shows a second order plant, with an input $x(t) = 5u(t)$ (Figure 5.21).

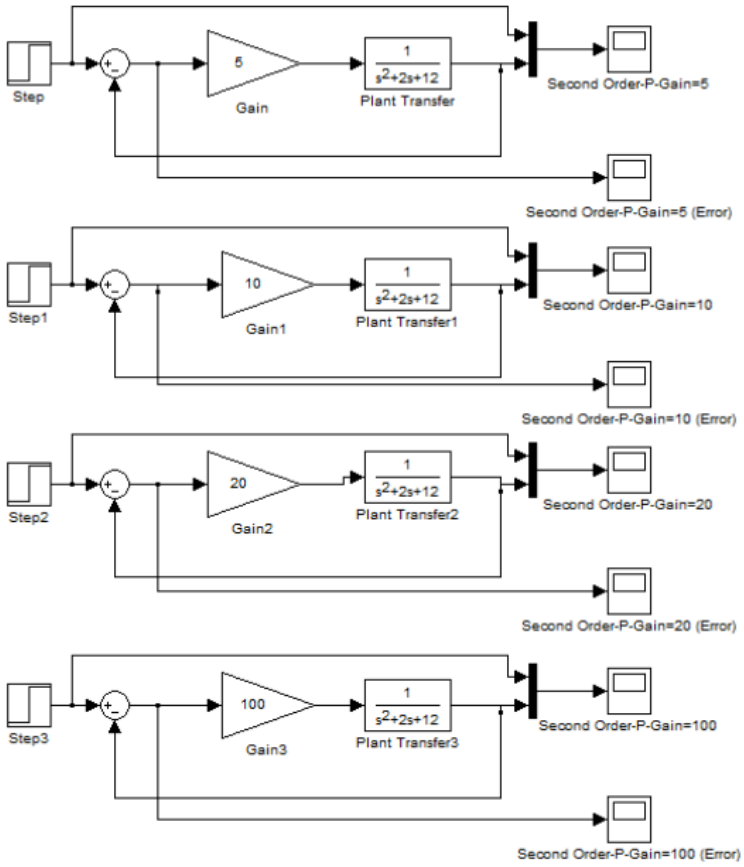


Figure 5.21: MATLAB-Simulink Diagram to show the effect of P control on second order plant

For Kp=5,

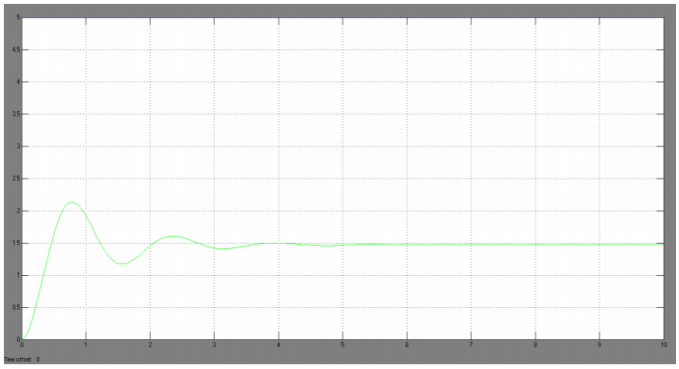


Figure 5.22: Output of the closed loop system with only P control, Kp=5

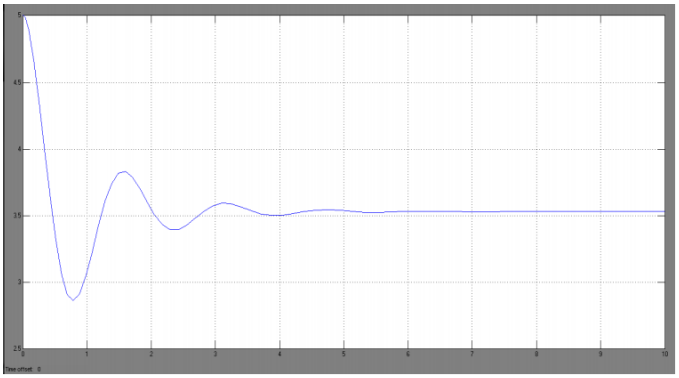


Figure 5.23: Error of the closed loop system with only P control, Kp=5

Steady State Error=$SSE = 3.5$. Percent Overshoot=43.4%
For Kp=10,

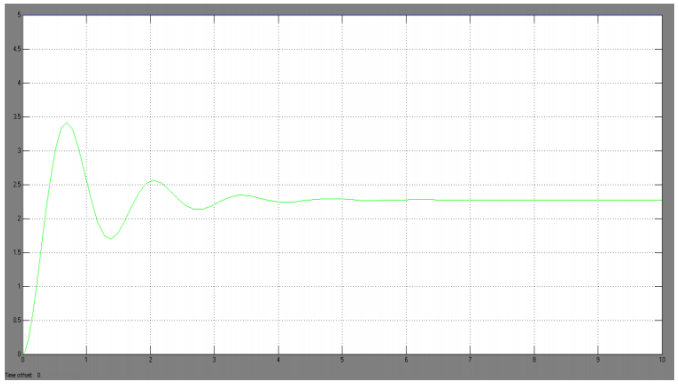


Figure 5.24: Output of the closed loop system with only P control, Kp=10

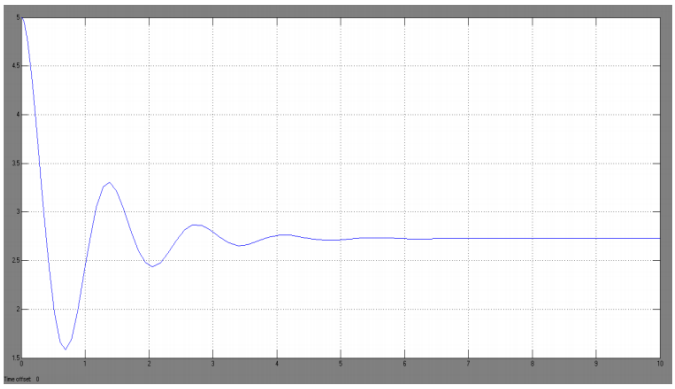


Figure 5.25: Error of the closed loop system with only P control, Kp=10

Steady State Error=$SSE = 2.7$. Percent Overshoot=47.83%
For Kp=20,

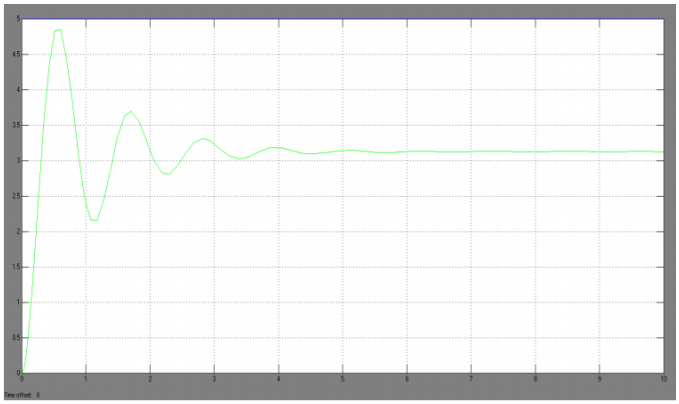


Figure 5.26: Output of the closed loop system with only P control, Kp=20

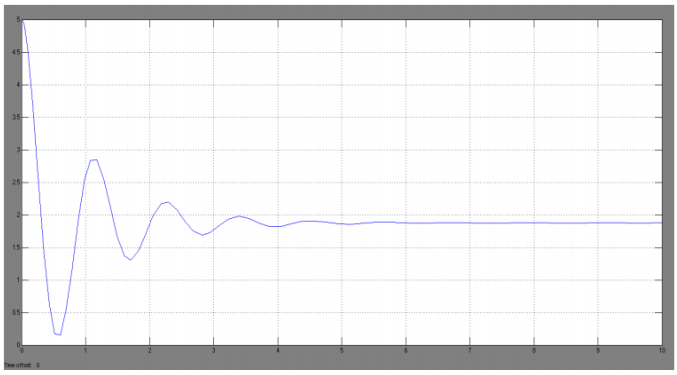


Figure 5.27: Error of the closed loop system with only P control, Kp=20

Steady State Error=$SSE = 1.9$. Percent Overshoot=54.84%
For Kp=100,

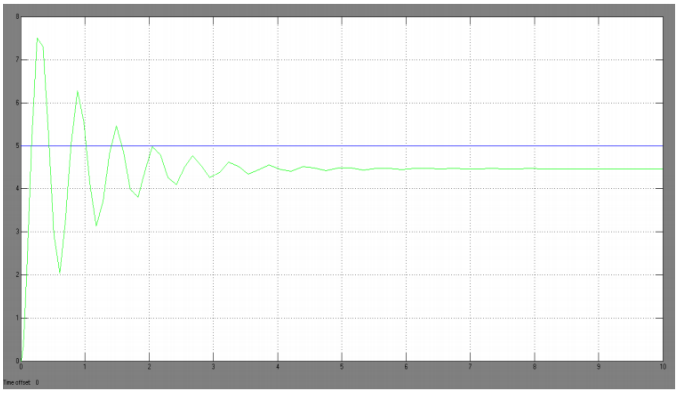


Figure 5.28: Output of the closed loop system with only P control, Kp=100

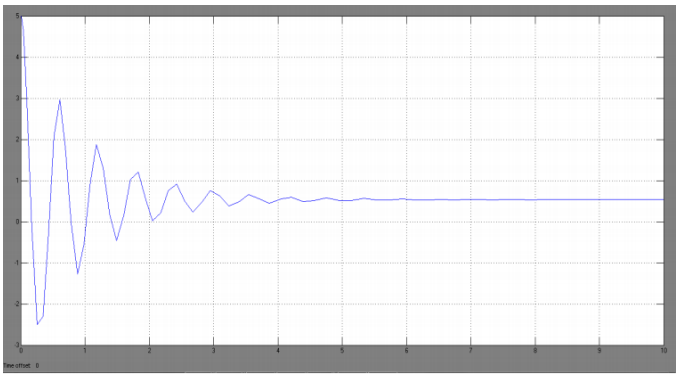


Figure 5.29: Error of the closed loop system with only P control, Kp=100

Steady State Error=$SSE = 0.5$. Percent Overshoot=66.67%

Transfer function: A Transfer Function is the ratio of the output of a system to the input of a system, in the Laplace domain considering its initial conditions and equilibrium point to be zero. The input of our system is the variation of the detected width (δm) and middle coordinate (δW) of the target, the output is the linear velocity (v) and the angular velocity (ω).

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} G_1(s) & 0 \\ 0 & G_2(s) \end{bmatrix} \begin{bmatrix} \delta m \\ \delta W \end{bmatrix}$$

To derive $G_1(s)$, we define θ as the change of the angle value, the corresponding arc length and chord length are, respectively, l and L; the radius is R. (Figure 5.30)

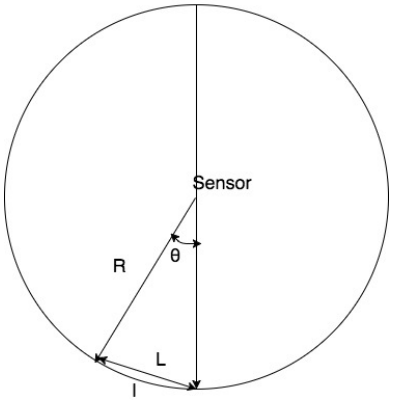


Figure 5.30: Diagram of the relevant parameter

$$\begin{cases} \omega = \frac{d\theta}{dt} \\ l = \theta * R \\ L = 2Rsin\frac{\theta}{2} \\ \delta m = L * k \end{cases} \longrightarrow \frac{d\delta m}{dt} = 2k * R * \omega cos\frac{\omega t}{2} \tag{5.1}$$

The Laplace transform is: (Note, to simplify, we assume that ω inside the $cos\frac{\omega t}{2}$ is a constant.)

$$\delta m(s) = 2kr\frac{\omega(s)}{s^2 + (\frac{\omega^2}{4})}$$

To derive the transfer function $G_2(s)$, we define m is the width of the target (arm to arm), d is the distance from the target to the sensor, θ is the corresponding angle which directly related to the detected width on the thermal image. (Figure 5.31)

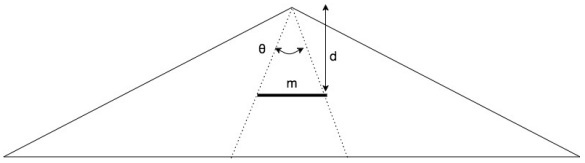


Figure 5.31: Diagram of the relevant parameter

$$\begin{matrix} m = 2d * tan\frac{\theta}{2} \approx \theta * d \\ \frac{\theta}{120^\circ} = \frac{W}{16[pixels]} \end{matrix} \longrightarrow m = 7.5w * d \tag{5.2}$$

We assume in two different frames, we have (W_1, d_1) and (W_2, d_2), we make a simple subtraction and obtain the following equation:

$$\delta W = W_1 - W_2 = \frac{m}{7.5} * \frac{\delta d}{d_1 * d_2}$$

To simplify, we now assume $d_1 * d_2$ is a constant, on both side we make the derivative of time:

$$\frac{d\delta W}{dt} = \frac{m}{7.5 * d_1 * d_2}\frac{d\delta d}{dt} = \frac{m}{7.5 * d_1 * d_2}v(t)$$

After Laplace transform, we obtain:

$$\delta W(s) = \frac{m}{7.5 * d_1 * d_2}\frac{1}{s}v(s)$$

Since the assumption of $d_1 * d_2 = constant$ is not correct, $d_1 * d_2$ is also time-dependent. In conclusion, neither $G_1(s)$ nor $G_2(s)$ are first order plants. Thus, we need to be careful with proportional control to avoid system oscillation.

From the example above, it is obvious that with a larger Kp, we reduce the Steady State Error, but increase the overshoot and the system starts oscillating before the steady state. To prevent the large overshoot, we could choose a smaller Kp, and set a limitation on output. In order to prevent the oscillation, we could set an "acceptable error range". The "acceptable error range" is a set by two threshold values. We modified the proportional control use the theory of bang-bang control for reference. The block diagram of bang-bang control is reported below (Figure 5.32):

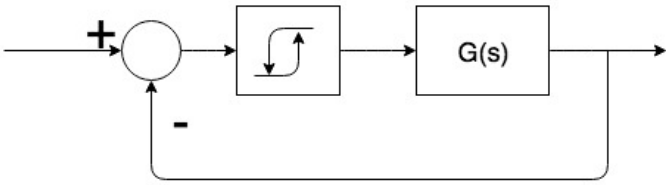


Figure 5.32: Bang-bang control diagram

As middle range we consider the columns from the 9^th (low threshold) to the 11^th (high threshold). When the middle point of the detected target contour is out of the middle range, the proportional control starts. When the middle point is inside the middle range, the proportional control stops.

5.4.4 Human Following

At the end of "Movement Estimation", the "Human Following" Node receives the data about the detected person: width of bounding rectangular, middle coordinate of the bounding rectangular and the state flag. Decide the reaction of the robot and send the velocity to the topic "cmd_vel", so

that the Node of the Robot Platform could receive the message and execute the command. Note, the "Human Following" Node here is not only a Publisher, but also a Subscriber.

The main loop can be subdivided in four steps:

- First the Listener roughly judge the Robot state basing on the received position flag.
- Analyse the received width of the contour and the middle coordinate of the contour, make a decision on the velocity.
- Send the velocity value to the topic "cmd_vel"
- The wheeled platform receives the velocity message and executes the command.

We defined four States: stop, searching, focusing and following. Next we will discuss the four states in detail.

stop: Every time we receive a "PositionInfo.flag" equals 0, the robot, switches to state "stop", namely, both the liner velocity (cmd_msg.linear.x) and the angular velocity (cmd_msg.angular.z) are set to zero, the robot stops. State "stop" exist under three conditions:

- The initial value of "PositionInfo.flag" is "0". It is obvious that the robot stops at the moment when it starts.
- Every time an "Intrude" appeals, the "PositionInfo.flag" is set to "0", the robot stops due to the confusion caused by the "Intrude".
- When the target person is in the proper middle area of the robot vision field, the robot stops.

searching: Every time we receive a "PositionInfo.flag" equals "2", means we lost the target. Thus we start searching the target. There are two conditions we defined of "lost target":

- The start moment of the robot, the "Target_Exist" flag is initialized to "None". If there is not a eligible target in the vision view, we consider it as "lost target".
- Once an "Intrude" appears, the robot enters the "stop" state and waits. If the person who caused the "Intrude" stays a long time in the vision field, the "Intrude_flag" can not be eliminated; we consider

it as "lost target". We consider it as "lost target" also when other conditions hold, e.g. the target person moves to another place while the person caused "Intrude" moving, as shown in Figure 5.33 and Figure 5.34.

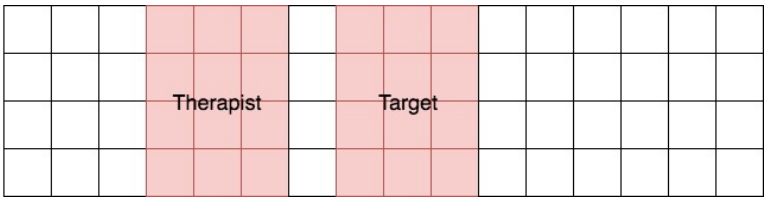


Figure 5.33: The therapist caused an Intrude

The therapist enters the vision field and has an comparable "hot range" as the target person. The "Intrudex_flag" is set to "1" due to the big change on the top-left coordinate of the contour.

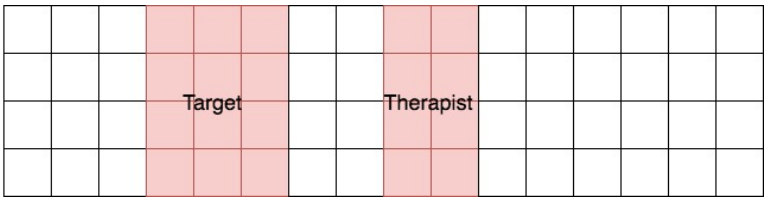


Figure 5.34: The confusion is still exist after the therapist passed as the target also moves during the waiting time.

The therapist walks to a further position while the target person moves to a new position, the big change on the top-left coordinate of the contour is still exist. The "Intrudex_flag" is not eliminated even the therapist is not an "Intrude" any more.

Under the first condition, the target person is not in the first frame but it is reasonable to assume the target person is not far from the robot, within 1 meter.

Under the second condition, considered within 2 seconds, the target person will not go too far away from the robot, we assume it is also within 1 meter.

Thus, during the "searching" state, we only set a angular velocity to make the robot turn around, in order to find the target person.

focusing and following: Position-Flag is "1", focusing is related to angular velocity and following is related to linear velocity. As we mentioned

before, we consider that the angular velocity and the linear velocity are two decoupled manipulated variables, the corresponding control variables are the middle coordinate and the width, respectively. Which means, the change of the middle coordinate is used to control the angular velocity while the change of the width used to control the linear velocity. (Figure 5.35)

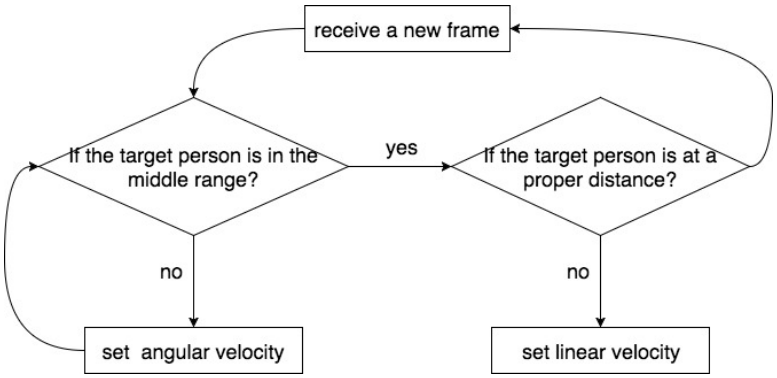


Figure 5.35: Diagram of velocity control

$$angularvelocity\omega = \delta m * Kp_1$$

$$linearvelocityv = \delta w * Kp_2$$

Where,

- $\delta m = target_m - 400$, target_m is the received middle coordinate of the target.
- $\delta w = target_w - 300$, target_w is the received width of the target.

Note, we multiplied 50 both on the column and the row during the thermal image conversion. Thus the middle coordinate is $8 * 50 = 400$ and the minimal acceptable width is $6 * 50 = 300$.
Figure 5.37 at end of this chapter shows the switching principle of the 4 states.

5.4.5 Interruption Check

During the debugging, we connect the Raspberry Pi with a keyboard and a screen to directly perceive the executing process. We use the keyboard to make an interruption in order to stop the main loop. Once we start testing the robot in real situations, there is no keyboard to stop the program, we

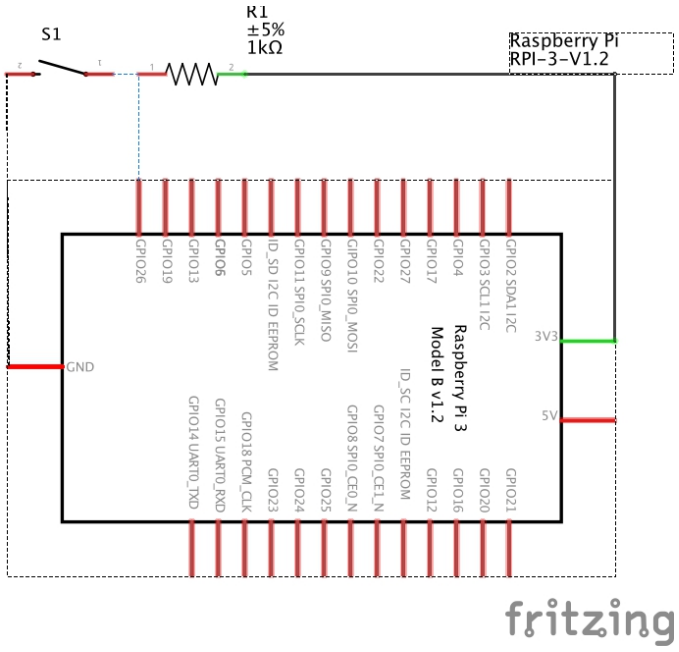


Figure 5.36: Circuit diagram of a push button

needed to add an interrupt button to do the same work. Figure 5.36 shows the circuit diagram of the interrupt button.

- Connect Pin 37 (GPIO 25) of the Raspberry Pi to one pin of the push button and one lead of the resistor.
- Connect the other end of the resistor to Pin 17 (3.3V) on Raspberry Pi.
- Connect the other pin of the push button to the ground.

Once the button is pressed, we receive a level jump at Pin 37, the main loop of the "target detection" process stops and a "stop" flag is send to the "human following" process.

```
input_value = GPIO.input(37)
if key == ord('q') or input_value == False:
    PositionInfo.flag = 0
    InfoPublisher.publish(PositionInfo)
    cv2.destroyAllWindows()
```

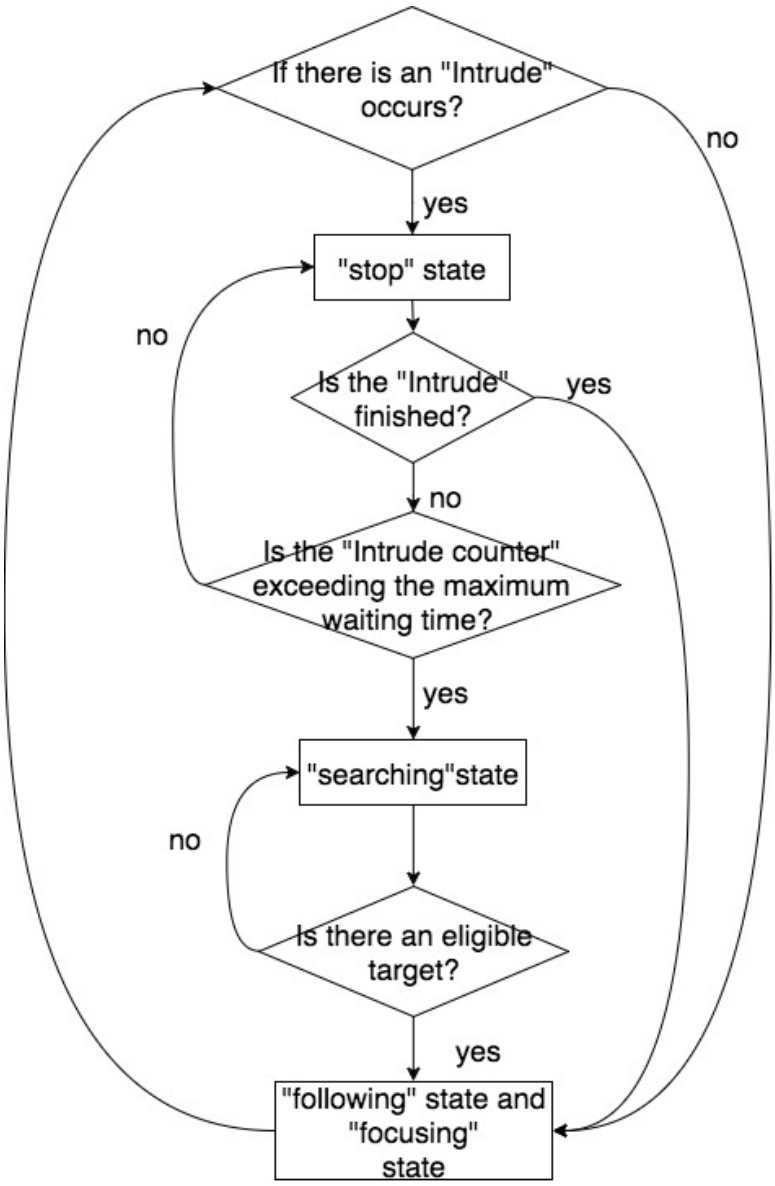


Figure 5.37: Diagram of the swiching between 4 states

Chapter 6

Experiments and Results

In this section, we created some experiments to test the algorithm and the effect of different aspects. To be better understand, we convert the temperature array into a color image. The color map we used to convert the temperature array to image is "JET" (Figure6.1). Thus the red range means the higher temperature detected (e.g., target person), the blue range means the lower temperature detected (e.g., background). The detected target range is marked in black frame.

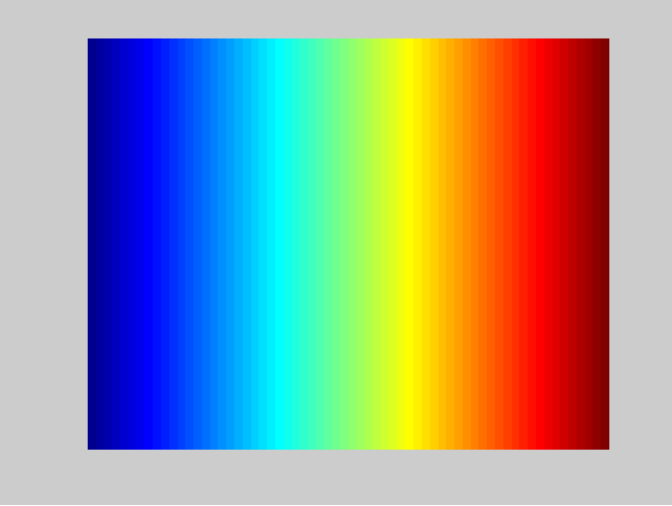


Figure 6.1: Color map "JET"

6.1 Effect of Clothes

The amount and color of clothing can make a small difference in sensor performance. Because clothing affects the amount of IR radiation emitted by the human body, it changes the differential between human body and ambient temperatures. With comparison of Figure 6.2 and Figure 6.3, the effect of color is minor compare to the ambient temperature.

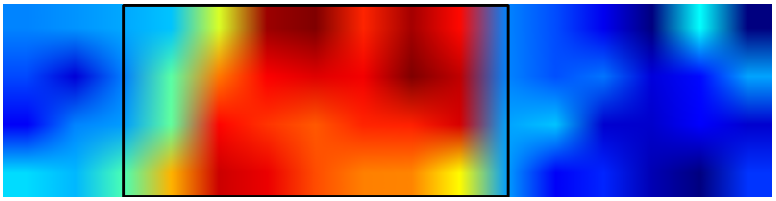


Figure 6.2: Target person wears white summer clothes

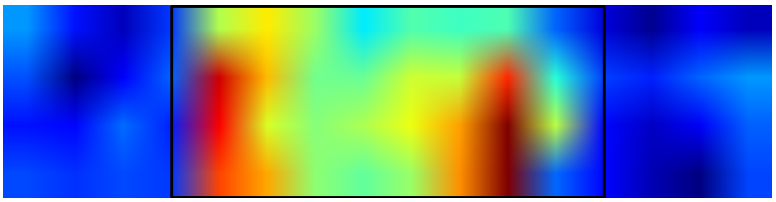


Figure 6.3: Target person wears black summer clothes

Figure 6.4 shows the condition when the target person is wearing winter coat. The sensor could hard distinguish the person from the background.

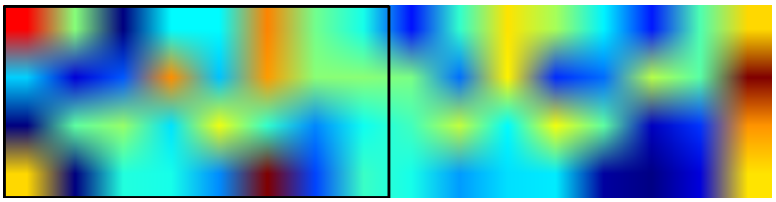


Figure 6.4: Target person wears heavy coat

6.2 Effect of Distance

The best IR sensors have a nominal sensing distance that can range from 2 to 12 meters, depending on the model. This distance specification does not necessarily represent the farthest distance the sensor can detect. Instead, it is the maximum distance at which sensing is guaranteed. In this section,

we compare the obtained image when the target person stands at different distance from the IR sensor. Figure 6.5 to Figure 6.8 is the target person wearing white summer clothes, and the width of the target person (arm to arm) is 45 cm.

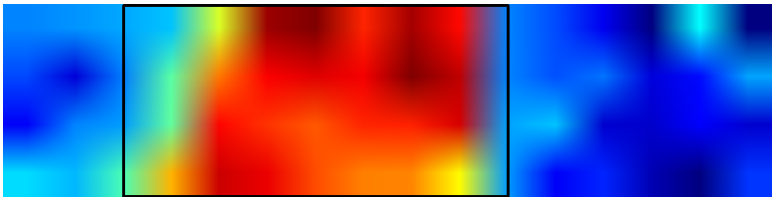


Figure 6.5: Target person wearing white summer cloths stands at a distance of 0.5 m

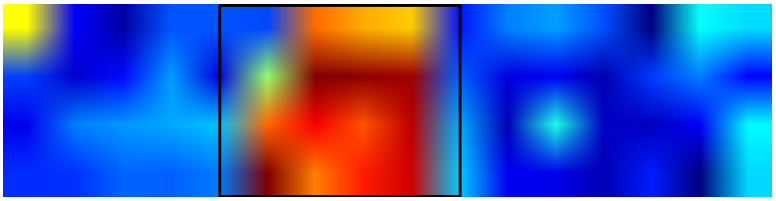


Figure 6.6: Target person wearing white summer cloths stands at a distance of 1.0 m

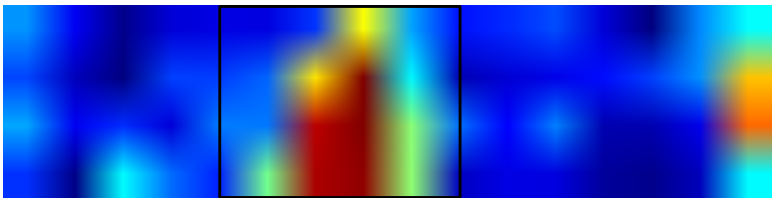


Figure 6.7: Target person wearing white summer cloths stands at a distance of 1.5 m

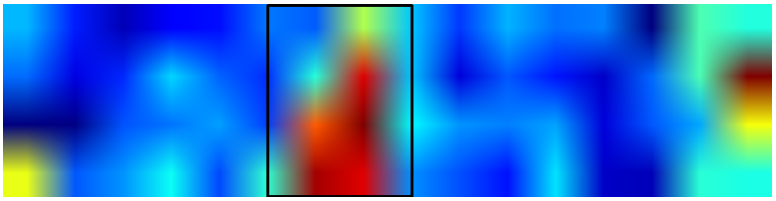


Figure 6.8: Target person wearing white summer cloths stands at a distance of 2.0 m

Figure6.9 to Figure6.13 is the target person wearing black summer clothes, and the width of the target person (arm to arm) is 50 cm.

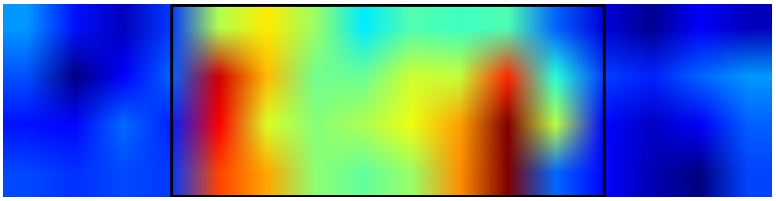


Figure 6.9: Target person wearing black summer cloths stands at a distance of 0.5 m

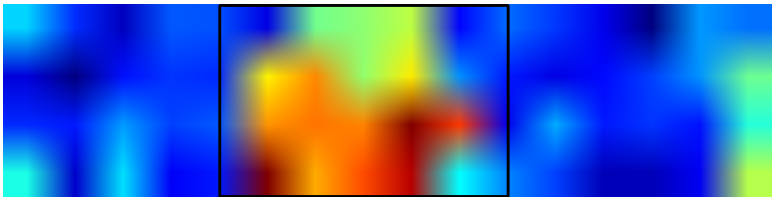


Figure 6.10: Target person wearing black summer cloths stands at a distance of 1.0 m

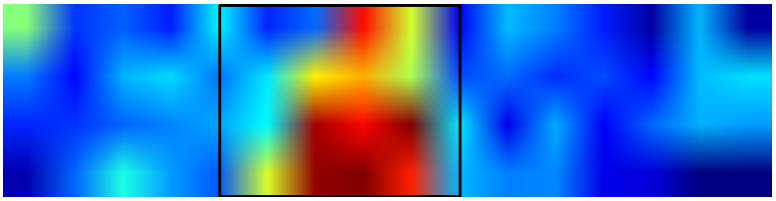


Figure 6.11: Target person wearing black summer cloths stands at a distance of 1.5 m

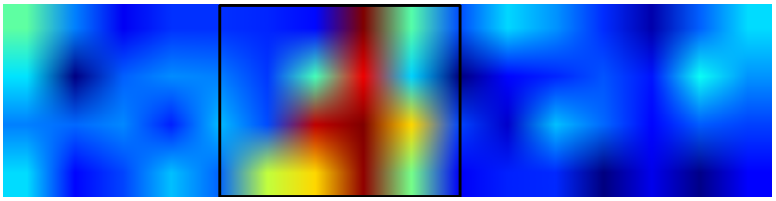


Figure 6.12: Target person wearing black summer cloths stands at a distance of 2.0 m

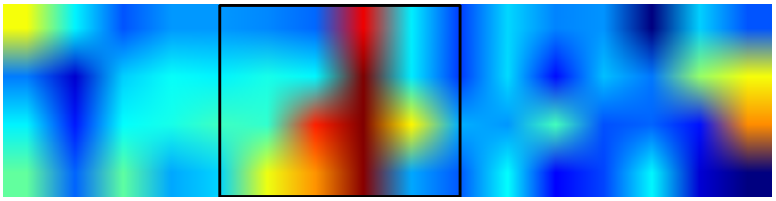


Figure 6.13: Target person wearing black summer cloths stands at a distance of 2.5 m

It is clear that as the distance increases, the resolution of the detected person decreases. And the effect of the color of clothes becomes greater after the distance increases over 1.5 m. The sensor is not able to estimate the distance changing of the target person wearing black clothes from Figure 6.11 to Figure 6.13

6.3 Effect of Intrude

In this section, the "Intrude" situation is tested by adding a moving person into the vision field. This person enters from right side (Figure6.14) and moves towards the target (Figure6.15). The person stops at the left side of the target and waits for a couple of seconds, causing confusion (Figure6.16). Then the person exits the vision field slowly. The confusion vanished after the maximum waiting time of "Intrude". The robot searches the target again and goes back to normal "human-following" process.

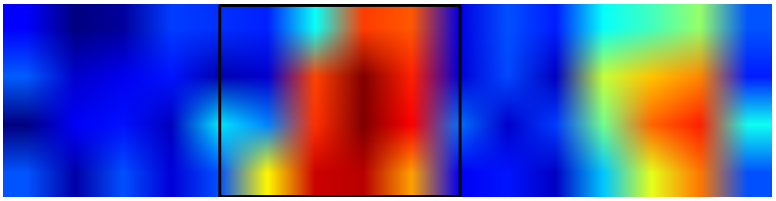


Figure 6.14: A person intrudes into the vision field from right side

The Target person is in the middle range of the thermal image. The pedestrian enters from right side. From the red color range we could figure out it is not hot enough compare to the target person. Thus it is still negligible.

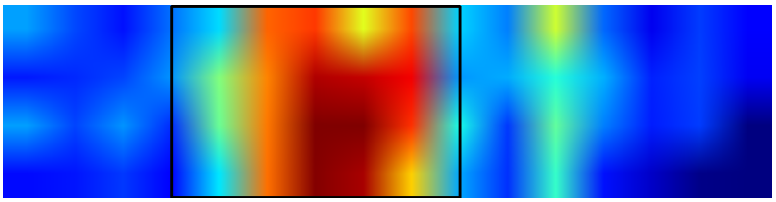


Figure 6.15: The person moves towards the target

As the pedestrian approaches the target person, the noise increases.

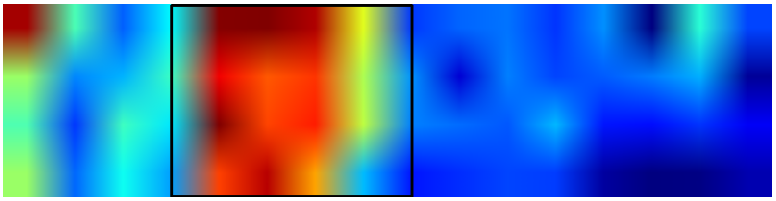


Figure 6.16: Intrude is generated

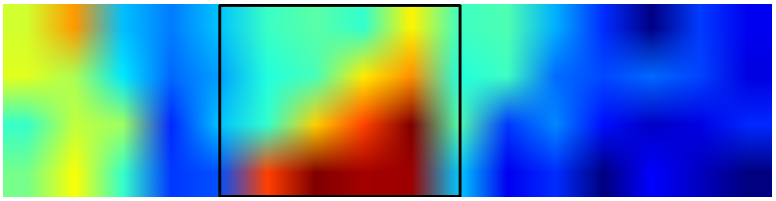


Figure 6.17: Keep the last result before "Intrude" and wait.

When the pedestrian and the target person overlapped in the thermal image, there is an big change of the width. An "Intrude" generated, we keep the former contour and wait.

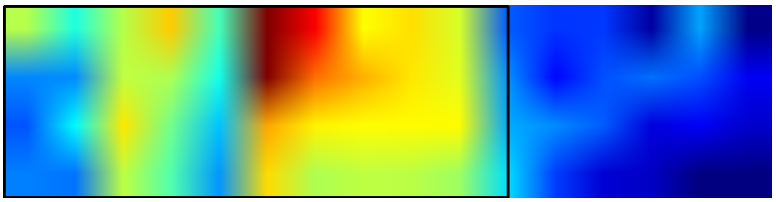


Figure 6.18: Problem occurs if the "Intrude" stays a long time.

The pedestrian stays for a long time and exceed the maximum waiting time of "Intrude". The "target detection" process consider that the target is lost and starts to find the new target. Since the pedestrian and the target person overlapped in the thermal image, a mistake occurs.

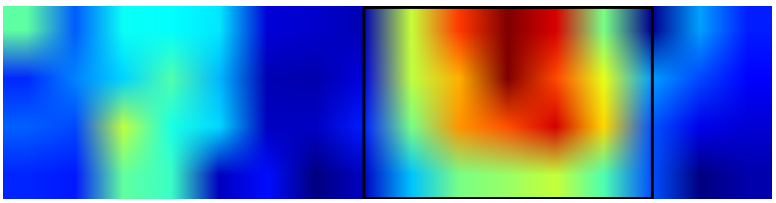


Figure 6.19: Searching for target again

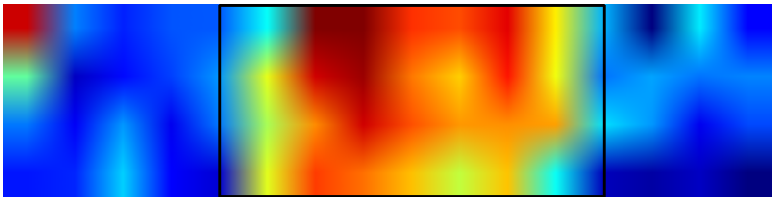


Figure 6.20: Back to normal tracking

After the pedestrian exits the vision field, there is another "Intrude" flag generated because of the big change on the width. And after the maximum waiting time the "target detection" process searches again the target and goes back to normal "human tracking" process again.

6.4 Effect of Room Temperature

In this section, the results of "human detection" under different room temperature conditions are given.

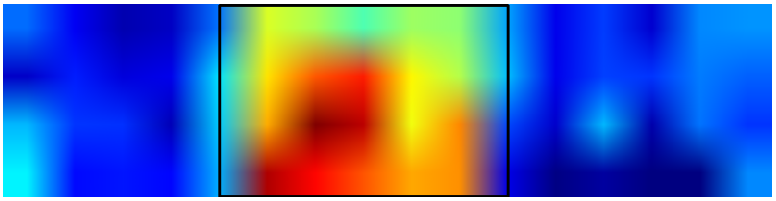


Figure 6.21: Room Temperature: 25° C

When the room temperature is 25°C, the target person and the background are clearly distinguished and form a sharp contrast.

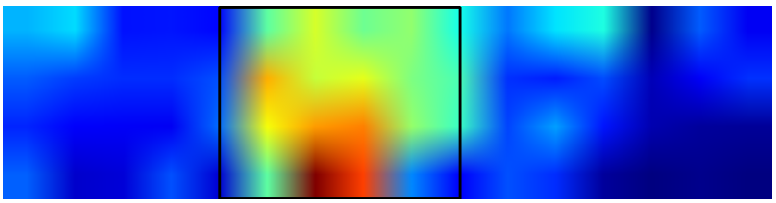


Figure 6.22: Room Temperature: 27° C

As the room temperature increases to 27°C, the target person and the background are still distinguished, but the contrast is less sharp than before.

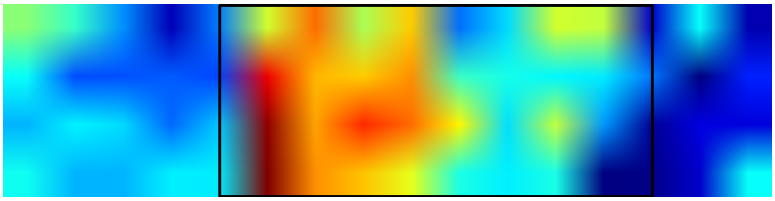


Figure 6.23: Room Temperature: 29° C

When the room temperature reaches 29°C, noise appears in the thermal image. The target person can not be clearly distinguished from the background.

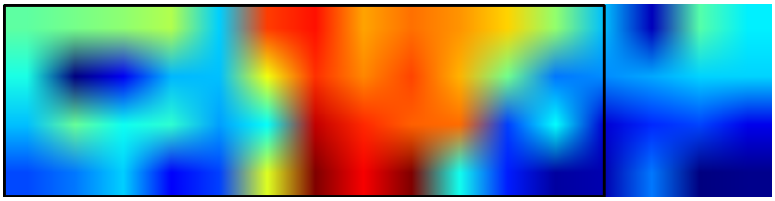


Figure 6.24: Room Temperature: 30° C

As the room temperature further increases, the noise increases. In conclusion, we suggest that the room temperature should be around 25°C.

6.5 Summary of Results

The robot is able to follow a person in a room without many other hot sources when the person walks slowly. The temperature acquisition process is the mainly determinant of the Frame Rate. If it is not set correctly, problems will occur in the following steps. The human detection process has some problems to detect a person that is wearing heavy clothes or a person that is very far. This involves that the user has to wear summer clothes and can not be further than 1.5 meter to the robot. The main problem of the human following process appears when another person enters the vision region and stays a long time. When this happens, the robot will receive an overlapping region of high temperature range, and finally considers there is only one person that stands close to the robot.

Chapter 7

Conclusion and Future work

7.1 Conclusion

In this paper, we have presented the design of control and IR sensor array based algorithm to detect and track people in indoor spaces from a mobile, holonomic platform. Traditional approaches to human-following typically involve a colour camera or a infrared camera, but this work aimed at a low-cost and reduced calculation approach for a specialized working environment condition (therapeutic room). This work has answered the following research questions.

- Is is possible to detect a person by using only a low-cost IR sensor array?
- Is it possible to follow a target person when the only information we have is the temperature range of the target person?
- If they are possible, are there any limitations of the algorithm?

This work presented that under an proper room temperature and without extra people walking around, the IR sensor array is able to detect a target person within 2 meters. Experimental results have shown that the target person must wear summer clothes and he/she is better detected with exposed arms or legs. The robot is able to follow the target person when the target person walks slowly.

We have implemented complete hardware and software setup to achieve the goal. The hardware setup includes infrared sensor platform and 3-omni wheels platform. The developed software includes mainly three parts: the first part calculates the temperature data and converts it to the thermal image. The second part consists in collecting data in order to obtain the

position information of the target person. The third part performs the analysis of the collected position data in order to compute correlated angular velocity and linear velocity of the robot.

7.2 Future Work

Although the goals were accomplished, the approach detailed in this thesis presents some aspects that can be implemented as future work.

First, by the time the project is finished, the new Melexis MLX90640 comes out, which is a low cost 32×24 pixels IR array. 768 FIR pixels could greatly improve the resolution of thermal image.

Second, depth information is not provided by using only a simple IR array. Lack of depth information could cause confusions when the target person is not facing the robot but looking sideways. The change of the width will be considered as a change of distance. By adding Kinect could provide us depth array to improve the accuracy of tracking.

Third, the algorithm sets strict limitations on working environment. Especially on the presentation of extra people, the algorithm can not distinguish the target person and the intruder if they are both close to the robot. By adding a camera could provide facial recognition when confusion is generated to make sure, we always follow the same target person.

Finally, obstacle avoidance could also be added by using ultrasonic sensors, which is much cheaper than other range data sensors.

Bibliography

[1] Björn F Andresen and Marija Strojnik. Infrared position sensitive detecter (irpsd). In *Infrared Technology and Applications XXIII*. SPIE, 1997.

[2] Andrea Bonarini, Matteo Matteucci, Martino Migliavacca, and Davide Rizzi. R2p: An open source hardware and software modular approach to robot prototyping. *Robotics and Autonomous Systems*, 62(7):1073–1084, 2014.

[3] Tech Collaborative. From internet to robotics, 2009.

[4] Terence Dwyer, James F Sallis, Leigh Blizzard, Ross Lazarus, and Kimberlie Dean. Relation of academic performance to physical activity and fitness in children. *Pediatric Exercise Science*, 13(3):225–237, 2001.

[5] Susan E Fasoli, Hermano I Krebs, Joel Stein, Walter R Frontera, and Neville Hogan. Effects of robotic therapy on motor impairment and recovery in chronic stroke. *Archives of physical medicine and rehabilitation*, 84(4):477–482, 2003.

[6] Ajo Fod, Andrew Howard, and MAJ Mataric. A laser-based people tracker. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 3, pages 3024–3029. IEEE, 2002.

[7] Takashi Hosono, Tomokazu Takahashi, Daisuke Deguchi, Ichiro Ide, Hiroshi Murase, Tomoyoshi Aizawa, and Masato Kawade. Human tracking using a far-infrared sensor array and a thermo-spatial sensitive histogram. In *Asian Conference on Computer Vision*, pages 262–274. Springer, 2014.

[8] V. KASSOVSKI, L. BUYDENS, and S. Maddalena. Infrared sensor with sensor temperature compensation, February 23 2016. US Patent 9,267,847.

[9] Massimo Piccardi. Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on*, volume 4, pages 3099–3104. IEEE, 2004.

[10] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B Cremers. People tracking with mobile robots using sample-based joint probabilistic data association filters. *The International Journal of Robotics Research*, 22(2):99–116, 2003.

[11] T.S. Villani. Uncooled ir detector array having improved temperature stability and reduced fixed pattern noise, June 24 2003. US Patent 6,583,416.

Appendix A

Installation of Ubuntu Mate and ROS

Install Ubuntu Mate on the SD card by following the instructions at

```
https://ubuntu-mate.org/raspberry-pi/
```

Take care of using an SD card with sufficient capacity for Ubuntu and ROS; at least 16 GB are suggested.
In order to install ROS Kinetic, which supports only Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04), we first setup the computer to accept software from packages ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
\$(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Set up keys

```
sudo apt-key dav --keyserver
hkp://ha.pool.sks-keyservers.net:80
--recv-key 421C365BD9FF1
F717815A3895523BAEEB01FA116
```

Make your Debian package index is up-to-date before installation.

```
sudo apt-get update
```

Desktop-Full Install: ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

```
sudo apt-get install ros-kinetic-desktop-full
```

Initialize rosdep to easily install system dependencies for source that is required to run some core components in ROS.

```
sudo rosdep init
rosdep update
```

Environment setup: It is convenient if the ROS environment variables are automatically added to bash session every time a new shell is launched.

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Getting rosinstall

```
sudo apt-get install python-rosinstall
```

Appendix B

Temperature Calculation Code Example (C++)

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>
#include <getopt.h>
#include <math.h>
#include <signal.h>
#include <bcm2835.h>

#define VERSION "0.1.0"
#define EXIT_FAILURE 1

char *xmalloc ();
char *xrealloc ();
char *xstrdup ();

float temperatures[64];
unsigned short temperaturesInt[64];
static int usage (int status);

/* The name the program was run with, stripped of any leading path. */
char *program_name;
```

```
/* getopt_long return codes */
enum {DUMMY_CODE=129
};

/* Option flags and variables */


static struct option const long_options[] =
{
  {"help", no_argument, 0, 'h'},
  {"version", no_argument, 0, 'V'},
  {NULL, 0, NULL, 0}
};

static int decode_switches (int argc, char **argv);
int mlx90621_init ();
int mlx90621_read_eeprom ();
int mlx90621_write_config (unsigned char *lsb, unsigned char *msb);
int mlx90621_read_config (unsigned char *lsb, unsigned char *msb);
int mlx90621_write_trim (char t);
char mlx90621_read_trim ();
int mlx90621_por ();
int mlx90621_set_refresh_hz (int hz);
int mlx90621_ptat ();
int mlx90621_cp ();
float mlx90621_ta ();
int mlx90621_ir_read ();
char EEPROM[256];
char ir_pixels[128];
char mlxFifo[] = "/var/run/mlx90621.sock";

void got_sigint(int sig) {
    unlink(mlxFifo);
    bcm2835_i2c_end();
    exit(0);

}

main (int argc, char **argv)
```

```
{
    signal(SIGINT, got_sigint);
    int fd;

    mkfifo(mlxFifo, 0666);

    int x;
    int i, j;
    float ta;
    int vir;
    int vcp;
    int acommon;
    int delta_ai;
    int delta_ai_scale;
    float epsilon;
    float ai;
    float bi;
    float vir_off_comp;
    float acp;
    float bcp;
    float vcp_off_comp;
    float tgc;
    float vir_tgc_comp;
    float vir_compensated;
    float ksta;
    int bi_scale;
    int alpha0;
    int alpha0_scale;
    int delta_alpha;
    int delta_alpha_scale;
    float alpha;
    float alphacp;
    float alpha_comp;
    int ks_scale;
    float ks;
    float tak;
    float sx;
    float to;

    program_name = argv[0];
```

```
i = decode_switches (argc, argv);

printf("\n");

if ( mlx90621_init() ) {
    printf("OK, MLX90621 init\n");
} else {
    printf("MLX90621 init failed!\n");
    exit(1);
}
ta = mlx90621_ta();
// If calibration fails then TA will be WAY too high.
// check and reinitialize if that happens
while (ta > 350)
{
    printf("Ta out of bounds! Max is 350, reading: %4.8f C\n", ta);
    //out of bounds, reset and check again
    mlx90621_init();
    ta = mlx90621_ta();
    usleep(10000);
}

printf("Ta = %f C %f F\n\n", ta, ta * (9.0/5.0) + 32.0);

/* To calc parameters */
unsigned char config_lsb, config_msb;
if ( !mlx90621_read_config( &config_lsb, &config_msb ) ) return 0;
float config_rg;
config_rg =config_lsb & 0b00110000;
config_rg = pow(2, (3.0 - config_rg));
vcp = mlx90621_cp();
epsilon = (( EEPROM[0xE5] << 8 ) | EEPROM[0xE4] ) / 32768.0;
acommon = ( EEPROM[0xD1] << 8 )| EEPROM[0xD0];
if(acommon>32767) acommon=acommon-65536;
delta_ai_scale = EEPROM[0xD9] & 0b11110000;
bi_scale = EEPROM[0xD9] & 0b00001111;
alpha0 = ( EEPROM[0xE1] << 8 ) | EEPROM[0xE0];
alpha0_scale = EEPROM[0xE2];
delta_alpha = EEPROM[0x80 + x];
```

```
delta_alpha_scale = EEPROM[0xE3];

/* do the work */
do {
    /* POR/Brown Out flag */

    while (!mlx90621_por) {
        sleep(1);
        mlx90620_init();
    }

    if ( !mlx90621_ir_read() ) exit(0);

for ( i = 0; i < 4; i++ ){
        for ( j = 0; j < 16; j++ ){
            x = ((j * 4) + i); /* index */

        /* OFFSET compensation vir, vcp */
        vir = ( ir_pixels[x*2+1] << 8 ) | ir_pixels[x*2];
        if(vir>32767) vir=vir-65536;
        delta_ai = (signed char)EEPROM[x];
        ai=(acommon+delta_ai*pow(2,delta_ai_scale))/config_rg;
        bi = EEPROM[0x40 + x];
        if(bi>127) bi=bi-256;
        bi = bi / (pow(2,bi_scale)*config_rg);
        vir_off_comp = vir - ( ai + bi * (ta - 25.0));
        acp = ( EEPROM[0xD4] << 8 )| EEPROM[0xD3];
        if(acp>32768) acp=acp-65536;
        acp = acp / config_rg;
        bcp = EEPROM[0xD5];
        if(bcp>127) bcp=bcp-256;
        bcp = bcp / (pow(2,bi_scale)*config_rg);
        vcp_off_comp = vcp - ( acp + bcp * (ta - 25.0));

        /* Thermal Gradient Compensation tgc */
        tgc = EEPROM[0xD8];
        if(tgc>127) tgc=tgc-256;
        tgc = tgc / 32.0;
        vir_tgc_comp = vir_off_comp - tgc * vcp_off_comp;
        //printf("vir_tgc_comp = %f\n", vir_tgc_comp);
```

```
            /* Emissivity compensation */
            vir_compensated = vir_tgc_comp / epsilon;

            /* Calculating alpha_comp */
            ksta=((EEPROM[0xE7]<<8)|EEPROM[0xE6]);
            if(ksta>127) ksta=ksta-256;
            ksta = ksta / 1048576.0;
            alpha=(alpha0/pow(2,alpha0_scale)
            +delta_alpha/pow(2,delta_alpha_scale))/config_rg;
            alphacp=(EEPROM[0xD7]<<8)|EEPROM[0xD6];
            alphacp=alphacp/(pow(2,alpha0_scale)*config_rg);
            alpha_comp=(1+ksta*(ta-25.0))*(alpha-tgc*alphacp);

            /*Calculating Ks*/
            ks_scale = EEPROM[0xC0] & 0b00001111;
            ks = EEPROM[0xC4];
            if(ks>127) ks=ks-256;
            ks = ks / pow(2,ks_scale+8);

            /*Calculation of to*/
            tak = pow((ta+273.15),4);
            sx=ks*pow((pow(alpha_comp,3)*vir_compensated
            +pow(alpha_comp,4)*tak),0.25);
            to=pow(vir_compensated
            /(alpha_comp*(1-ks*273.15)+sx)+tak,0.25)-273.15;
            temperaturesInt[x] = (unsigned short)((to + 273.15) * 100.0)
            //give back as Kelvin (hundtredths of degree)
            temperatures[x] = to;

            }
    }

        if((fd = open(mlxFifo, O_WRONLY | O_CREAT)) == -1) {
        printf("Open Error");
        exit(1);
    }

        write(fd, temperaturesInt, sizeof(temperaturesInt));
        close(fd);
```

```
        printf("Updated Temperatures!\n");
        usleep(10000);

    } while (1);

    unlink(mlxFifo);

    exit (0);
}


/* Init */

int
mlx90621_init()
{
    if (!bcm2835_init()) return 0;
    bcm2835_i2c_begin();
    bcm2835_i2c_set_baudrate(25000);

    //sleep 5ms per datasheet
    usleep(5000);
    if ( !mlx90621_read_eeprom() )
        return 0;
    if ( !mlx90621_write_trim( EEPROM[0xF7] ) )
        return 0;
    if ( !mlx90621_write_config( &EEPROM[0xF5], &EEPROM[0xF6] ) )
        return 0;

    mlx90621_set_refresh_hz( 16 );
    unsigned char lsb, msb;
    mlx90621_read_config( &lsb, &msb );
    return 1;
}

/* Read the whole EEPROM */

int
mlx90621_read_eeprom()
{
```

```
    const unsigned char read_eeprom[] = {
        0x00 // command
    };

    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x50);
    if (
        bcm2835_i2c_write_read_rs(
        (char *)&read_eeprom, 1, EEPROM, 256)
        == BCM2835_I2C_REASON_OK
        ) return 1;

    return 0;
}

/* Write device configuration value */

int
mlx90621_write_config(unsigned char *lsb, unsigned char *msb)
{
    unsigned char lsb_check = lsb[0] - 0x55;
    unsigned char msb_check = msb[0] - 0x55;

    unsigned char write_config[] = {
        0x03, // command
        lsb_check,
        lsb[0],
        msb_check,
        msb[0]
    };

    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x60);
    if (
        bcm2835_i2c_write((const char *)&write_config, 5)
        == BCM2835_I2C_REASON_OK
        ) return 1;

    return 0;
}
```

```
/* Reading configuration */

int
mlx90621_read_config(unsigned char *lsb, unsigned char *msb)
{
    unsigned char config[2];

    const unsigned char read_config[] = {
        0x02, // command
        0x92, // start address
        0x00, // address step
        0x01  // number of reads
    };

    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x60);
    if (
        !bcm2835_i2c_write_read_rs((char *)&read_config, 4, config, 2)
        == BCM2835_I2C_REASON_OK
        ) return 0;

    *lsb = config[0];
    *msb = config[1];
    return 1;
}

/* Write the oscillator trimming value */

int
mlx90621_write_trim(char t)
{
    unsigned char trim[] = {
        0x00, // MSB
        t     // LSB
    };
    unsigned char trim_check_lsb = trim[1] - 0xAA;
    unsigned char trim_check_msb = trim[0] - 0xAA;

    unsigned char write_trim[] = {
```

```
        0x04, // command
        trim_check_lsb,
        trim[1],
        trim_check_msb,
        trim[0]
    };

    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x60);
    if (
        bcm2835_i2c_write((char *)&write_trim, 5)
        == BCM2835_I2C_REASON_OK
        ) return 1;

    return 0;
}

/* Read oscillator trimming register */

char
mlx90621_read_trim()
{
    unsigned char trim_bytes[2];

    const unsigned char read_trim[] = {
        0x02, // command
        0x93, // start address
        0x00, // address step
        0x01  // number of reads
    };

    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x60);
    if (
        bcm2835_i2c_write_read_rs((char *)&read_trim, 4, trim_bytes, 2)
        == BCM2835_I2C_REASON_OK
        ) return 1;

    return trim_bytes[0];
}
```

```
/* Return POR/Brown-out flag */

int
mlx90621_por()
{
    unsigned char config_lsb, config_msb;
    mlx90621_read_config( &config_lsb, &config_msb );
    return ((config_msb & 0x04) == 0x04);
}

/* Set IR Refresh rate */

int
mlx90621_set_refresh_hz(int hz)
{
    char rate_bits;

    switch (hz) {
        case 512:
            rate_bits = 0b0000;
            break;
        case 256:
            rate_bits = 0b0110;
            break;
        case 128:
            rate_bits = 0b0111;
            break;
        case 64:
            rate_bits = 0b1000;
            break;
        case 32:
            rate_bits = 0b1001;
            break;
        case 16:
            rate_bits = 0b1010;
            break;
        case 8:
            rate_bits = 0b1011;
            break;
```

```
        case 4:
            rate_bits = 0b1100;
            break;
        case 2:
            rate_bits = 0b1101;
            break;
        case 1:
            rate_bits = 0b1110; // default
            break;
        case 0:
            rate_bits = 0b1111; // 0.5 Hz
            break;
        default:
            rate_bits = 0b1110;
    }

    unsigned char config_lsb, config_msb;
    if ( !mlx90621_read_config( &config_lsb, &config_msb ) )
        return 0;
    config_lsb = rate_bits;
    if ( !mlx90621_write_config( &config_lsb, &config_msb ) )
        return 0;

    return 1;
}

/* Return PTAT (Proportional To Absolute Temperature) */

int
mlx90621_ptat()
{
    int ptat;
    unsigned char ptat_bytes[2];

    const unsigned char read_ptat[] = {
        0x02, // command
        0x40, // start address
        0x00, // address step
        0x01  // number of reads
    };
```

```
    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x60);
    if (
        !bcm2835_i2c_write_read_rs((char *)
        &read_ptat, 4, (char *)&ptat_bytes, 2)
        == BCM2835_I2C_REASON_OK
        ) return 0;

    ptat = ( ptat_bytes[1] << 8 ) | ptat_bytes[0];
    return ptat;
}

/* Compensation pixel read */

int
mlx90621_cp()
{
    int cp;
    signed char VCP_BYTES[2];

    const unsigned char compensation_pixel_read[] = {
        0x02, // command
        0x41, // start address
        0x00, // address step
        0x01  // number of reads
    };

    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x60);
    if (
        !bcm2835_i2c_write_read_rs((char*)
        &compensation_pixel_read,4,(char*)&VCP_BYTES,2)
        == BCM2835_I2C_REASON_OK
        ) return 0;

    cp = ( VCP_BYTES[1] << 8 ) | VCP_BYTES[0];
    return cp;
}
```

```
/* calculation of absolute chip temperature */

float
mlx90621_ta()
{
    int ptat = mlx90621_ptat();
    unsigned char config_lsb, config_msb;
    if ( !mlx90621_read_config( &config_lsb, &config_msb ) )
        return 0;
    float config_rg;
    config_rg =config_lsb & 0b00110000;
    config_rg = pow(2, (3 - config_rg));
    float vth = (( EEPROM[0xDB] << 8 ) | EEPROM[0xDA]);
    if (vth > 32767) vth = vth - 65536;
    vth= vth/config_rg;

    float kt1 = (( EEPROM[0xDD] << 8 ) | EEPROM[0xDC]);
    int tem_kt1 = EEPROM[0xD2] & 0b11110000;
    tem_kt1 = tem_kt1 >> 4;
    tem_kt1 = pow(2, tem_kt1)*config_rg;
    kt1 = kt1/tem_kt1;

    float kt2 = (( EEPROM[0xDF] << 8 ) | EEPROM[0xDE]);
    int tem_kt2 = EEPROM[0xD2] & 0b00001111;
    tem_kt2 = pow(2, (tem_kt2 + 0x0A))*config_rg;
    kt2 = kt2/tem_kt2;

    float ta=((-kt1+sqrt( kt1*kt1-(4 * kt2)*(vth-ptat)))/(2*kt2))+25;
    printf("%d %f %f %f\n",ptat, vth, kt1, kt2);

    return ta;
}

/* IR data read */

int
mlx90621_ir_read()
{
    const unsigned char ir_whole_frame_read[] = {
        0x02, // command
```

```
        0x00, // start address
        0x01, // address step
        0x40  // number of reads
    };

    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x60);
    if (
        bcm2835_i2c_write_read_rs((char *)
        &ir_whole_frame_read, 4, ir_pixels, 128)
        == BCM2835_I2C_REASON_OK
        ) return 1;

    return 0;
}


/* Set all the option flags according to the switches specified.
   Return the index of the first non-option argument.
*/

static int
decode_switches (int argc, char **argv)
{
  int c;


  while ((c = getopt_long (argc, argv,
                "h"  /* help */
                "V", /* version */
                long_options, (int *) 0)) != EOF)
    {
      switch (c)
    {
    case 'V':
      printf ("mlx %s\n", VERSION);
      exit (0);

    case 'h':
      usage (0);
```

```
    default:
      usage (EXIT_FAILURE);
    }
    }

  return optind;
}


static int
usage (int status)
{
  printf ("%s - \
\n", program_name);
  printf ("Usage: %s [OPTION]... [FILE]...\n", program_name);
  printf ("\
Options:\n\
  -h, --help                 display this help and exit\n\
  -V, --version              output version information and exit\n\
");
  exit (status);
}
```

Appendix C

Human Detection Node Code Example (Python)

```
#!/usr/bin/env python
# license removed for brevity
#import rospy to write a ros Node
import rospy
# import related libraries
import time
import math
import numpy as np
import subprocess
import os, sys
import skimage
from skimage import io,exposure,transform,img_as_float,img_as_ubyte
from time import sleep
import cv2
import RPi.GPIO as GPIO
from beginner_tutorials.msg import IntList
from std_msgs.msg import Int32
# Initialization of important values and flags
GPIO.setmode(GPIO.BOARD)
GPIO.setup(37,GPIO.IN)
fifo= open('/var/run/mlx90621.sock', 'r')
Target_Exist = None
Intrudew_flag = None
Intrudex_flag = None
Intrudew_count= 0
```

```
Intrudex_count= 0
y_target = None
x_target = 0
w_target = 0
InfoPublisher = rospy.Publisher('Info', IntList, queue_size = 10)
rospy.init_node('IRSensorPublisher', anonymous=True)
PositionInfo = IntList()
PositionInfo.flag = 0
PositionInfo.w = 800
PositionInfo.m = 800

#update loop
while not rospy.is_shutdown():
    while Target_Exist is None:
        sleep(0.08)
        ir_raw= fifo.read()
        ir_trimmed= ir_raw[0:128]
        ir= np.frombuffer(ir_trimmed, np.uint16)
# check if there is a person by checking how many pixels over 28degree
        T30 = 0
        for pixel in ir:
            if pixel>30315:
                T30 = T30+1
        if T30 < 8:
            Target_Exist = None
            if Intrudew_count==0 or Intrudex_count==0:
                PositionInfo.flag = 0
                InfoPublisher.publish(PositionInfo)
                Intrudew_count = None
                Intrudex_count = None
                stop_count = 0
            else:
                stop_count = stop_count+1
                if stop_count ==20:
                    PositionInfo.flag = 2
                    InfoPublisher.publish(PositionInfo)
                    stop_count = 0
        else:
            Target_Exist = 1
            PositionInfo.flag = 1
```

```
    #An eligible target is found.
    sleep(0.06)
    ir_raw= fifo.read()
    ir_trimmed= ir_raw[0:128]
    ir= np.frombuffer(ir_trimmed, np.uint16)
    ir= ir.reshape((16,4))[::-1, ::-1]
    ir= img_as_float(ir)
    p2, p80, p98 = np.percentile(ir, (2, 80, 98))
    th = np.uint8(p80 * 255)
    if p2 == p98:
        print ('p2 = p98')
    else:
        ir = exposure.rescale_intensity(ir, in_range=(p2, p98))
    # Convert to color image
    rgba_img = ir * 255.0
    rgba_img = rgba_img.astype(np.uint8)
    rgba_img = np.rot90(rgba_img, 3)
    rgb_img = cv2.applyColorMap(rgba_img, cv2.COLORMAP_JET)
    thresh = cv2.threshold(rgba_img, th, 255, cv2.THRESH_BINARY)[1]
    row, col=rgba_img.shape[:2]
    rgb_img=cv2.resize(rgb_img, (50*col, 50*row),
interpolation = cv2.INTER_LINEAR)
    rgba_img=cv2.resize(rgba_img, (50*col, 50*row),
    interpolation = cv2.INTER_LINEAR)
    thresh=cv2.resize(thresh, (50*col, 50*row),
    interpolation = cv2.INTER_LINEAR)

    if Target_Exist == 1:
        # find contours
        (image, cnts, _) = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        xs = np.zeros(10, dtype = np.uint16)
        ys = np.zeros(10, dtype = np.uint16)
        ws = np.zeros(10, dtype = np.uint16)
        hs = np.zeros(10, dtype = np.uint16)
        delta = np.zeros(10, dtype = np.uint16)
        Area = np.zeros(10, dtype = np.uint16)
        i = 0
        # loop over the contours
        for c in cnts:
```

```
        # if the contour is too small, ignore it
            if cv2.contourArea(c) < 1500:
                continue
# For contours large enough, we just choose the biggest one
            else:
                Area[i] = cv2.contourArea(c)
                (x,y,w,h) =  cv2.boundingRect(c)
                xs[i] = np.uint16(x)
                ys[i] = np.uint16(y)
                ws[i] = np.uint16(w)
                hs[i] = np.uint16(h)
                i = i + 1
    # now we choose the biggest contour and draw the rectangle
        index = np.argmax(Area)
        if (abs(np.float(x_target) - np.float(xs[index])) < 200)
        or (y_target == None) :
            Intrudex_flag = None
            if (abs(np.float(ws[index]) - np.float(w_target)) < 200)
            or (y_target == None) :
                Intrudew_flag = None
                x_target = xs[index]
                y_target = ys[index]
                w_target = ws[index]
                h_target = hs[index]
            else:
                x_target = xs[index]
                y_target = ys[index]
                h_target = hs[index]
                if Intrudew_flag == None:
                    Intrudew_flag = 1
                    Intrudew_count = 1
                else:
                    if Intrudew_count < 40:
                        Intrudew_count = Intrudew_count + 1
                    else:
                        Intrudew_flag = None
                        Intrudew_count = 0
                        y_target = None
                        x_target = 0
                        w_target = 0
```

```
                    Target_Exist = None
                    PositionInfo.flag = 0
                    InfoPublisher.publish(PositionInfo)
                    continue
        else:
            if Intrudex_flag == None:
                Intrudex_flag = 1
                Intrudex_count = 1
            else:
                if Intrudex_count < 40:
                    Intrudex_count = Intrudex_count + 1
                else:
                    Intrudex_flag = None
                    Intrudex_count = 0
                    y_target = None
                    x_target = 0
                    w_target = 0
                    Target_Exist = None
                    PositionInfo.flag = 0
                    InfoPublisher.publish(PositionInfo)
                    continue
    # Draw the contour rectangle and display the thermal image
        cv2.rectangle(rgb_img, (x_target,y_target),
        (x_target+w_target, y_target+h_target), (0,0,0), 2)
        cv2.imshow('Thresh', thresh)
        cv2.imshow('gray', rgba_img)
        cv2.imshow('tracking', rgb_img)
     # Compute the important characteristcs of the contour
        temp_m = np.array([np.float(w_target/2 + x_target)])
        temp_w = np.array([w_target])
        PositionInfo.m = int(np.asscalar(temp_m))
        PositionInfo.w = np.asscalar(temp_w)
        InfoPublisher.publish(PositionInfo)
        print("m="+str(PositionInfo.m),"w="+str(PositionInfo.w))
    # if an interruption happens, break from the loop
        key = cv2.waitKey(1) & 0xFF
        input_value = GPIO.input(37)
        if key == ord('q') or input_value == True:
            PositionInfo.flag = 0
            InfoPublisher.publish(PositionInfo)
```

```
                cv2.destroyAllWindows()
                print('Waiting...')
                # wait until the button pressed again
                sleep(1)
                input_value = GPIO.input(37)
                while input_value == False:
                    PositionInfo.flag = 0
                    InfoPublisher.publish(PositionInfo)
                    input_value = GPIO.input(37)
                #GPIO.wait_for_edge(37,GPIO.RISING)
                Target_Exist = None
                Intrudew_flag = None
                Intrudex_flag = None
                Intrudew_count= 0
                Intrudex_count= 0
                y_target = None
                x_target = 0
                w_target = 0
                PositionInfo.flag = 0
                PositionInfo.w = 800
                PositionInfo.m = 800
                continue
                # break
            else:
                continue
cv2.destroyAllWindows()
print('Closing...')
fifo.close()
```

Appendix D

Control Theory Code Example

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include "beginner_tutorials/IntList.h"
#include <stdlib.h>
enum State
    {
        stop,searching,focusing,following
    };
State state;
int width;
int middle;

void positionCallback(const beginner_tutorials::IntList& position_msg)
{

    if(position_msg.flag == 0)
    {
        state = stop;
    }
    else if(position_msg.flag == 2)
    {
        state = searching;
    }
    else if(position_msg.flag == 4)
    {
```

```
        exit(0);
    }
    else
    {
        width = int(position_msg.w);
        middle = int(position_msg.m);
        if(middle<350 || middle>450)
        {
            state = focusing;
        }
        else
            state = following;
    }
}


void stopState(geometry_msgs::Twist& cmd_msg)
{
    cmd_msg.linear.x = 0;
    cmd_msg.angular.z = 0;
}

void searchingState(geometry_msgs::Twist& cmd_msg)
{
    cmd_msg.linear.x = 0;
    cmd_msg.angular.z = 1.0;
}

void focusingState(geometry_msgs::Twist& cmd_msg)
{
    float Delta_m = middle*1.0 - 400.0;
    float angular = -0.005*Delta_m;
    if (angular<-1.0)
        {angular = -1.0;}
    if (angular>1.0)
        {angular =1.0;}
    cmd_msg.angular.z = angular;
    cmd_msg.linear.x = 0;
}
```

```
void followingState(geometry_msgs::Twist& cmd_msg)
{

    if (width<=400 && width>300)
    {
        cmd_msg.linear.x = 0;
        cmd_msg.angular.z = 0;
    }
    else
    {
        float Delta_w = width*1.0 - 350.0;
        float velocity = -0.004*Delta_w;
        if(velocity<-1.0)
            {velocity=-1.0;}
        if(velocity>1.0)
            {velocity=1.0;}
        cmd_msg.linear.x = velocity;
        cmd_msg.angular.z = 0;
    }
}

void stateTransition(geometry_msgs::Twist& cmd_msg)
{

    switch(state)
    {
        case stop:
            stopState(cmd_msg);
            break;
        case searching:
            searchingState(cmd_msg);
            break;
        case following:
            followingState(cmd_msg);
            break;
        case focusing:
            focusingState(cmd_msg);
            break;
    }
}
```

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "trategy");
    ros::NodeHandle n;
    ros::Rate rate(17);
    ros::Subscriber sub=n.subscribe("Info",10, positionCallback);
    ros::Publisher cmd_pub=n.advertise<geometry_msgs::Twist>
    ("/cmd_vel",5);
    geometry_msgs::Twist cmd_msg;
    while(ros::ok())
    {
        ros::spinOnce();
        stateTransition(cmd_msg);
        cmd_pub.publish(cmd_msg);
        rate.sleep();
    }
    return 0;
}
```

Appendix E

Glossary

ADC Analog to Digital Converter. 20
BSD Berkeley Software Distribution. 24
CCD Charge Coupled Device. 21
DCM DC Motor controller board. 49
FIFO First In, First Out. 41, 56-58
FOV Field Of View. 28-30. 38, 57, 65-66
FPN Fixed-Pattern Noise. 36
GPIO General Purpose Input Output. 23, 31, 32, 64, 78
I2C Inter-Integrated Circuit. 23, 28, 32, 33
IMU Inertial Measurement Unit. 49
MCU Microprogrammed Control Unit. 35
NETD Noise Equivalent Temperature Difference. 29
PTAT Proportional To Absolute Temperature. 28, 36
POR Power On Reset. 35
PS Power Supply. 49
PWM Pulse Width Modulation. 63, 64
RAM Random Access Memory. 22
ROS Robot Operative System. 23-25, 49
RPC Remote Procedure Call. 24
SCL Serial Clock Input. 31, 32
SDA Serial Data. 31, 32
SPI Serial Peripheral Interface. 23
SSE Steady State Error. 68-72
TGC Thermal Gradient Compensation. 36, 37
WMR Wheeled Mobile Robot. 42