University of Southampton

Faculty of Engineering and Physical Sciences

Autonomous guidance system design for a low cost All-Terrain Robot base on fiducial markers

By

Ping-Chun Tsai

Pct1g19@soton.ac.uk

OCT 2020

Supervisor: Dr. Klaus-Peter

Second Examiner: Danesh S Tarapore

A dissertation submitted in partial fulfilment of the degree of MSc Artificial intelligence

A dissertation submitted in partial fulfilment of the degree of MSc Artificial intelligence

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Acknowledgement

I would like to thank my supervisors, Dr. Klaus-Peter Zauner for his great support on project direction and code development advice throughout the research project. I would like to express my appreciation towards my second supervisor Danesh S Tarapore for giving me feedback on project demonstration.

University of Southampton

Faculty of Engineering and Physical Sciences

Abstract

In embedded computer vision field, a guidance system for all-terrain robot is very crucial. This project aims to design a computer vision based guidance system with the opensource software and hardware. The purpose for this guidance system is to satisfy the teaching requirements used in the ELEC2209 course at Southampton University. The developed guidance system has the advantageous features of low-cost and robustness under different environments. The guidance system is designed as the upper layer compute unit in the robot system. The expected three outputs from the guidance system are: The GPIO controller pin output, UART Serial communication I/O and the wireless access point interface. To validate the guidance system design, several experiments are conducted under various lighting conditions.

Table of Content

1.Introduction：

The guidance system plays important role in the autonomous robotic system. Especially in the human unreachable environment or communication transmit is difficult environment. The embedded computer vision system is one of the main subjects in the system. The computer vision can monitor the roads condition and surround environment. Any embedded system you want to use for autonomous computer vision applications will need run at minimum trimmed down version of standard OS and standard libris for machine learning models, it can program into FPGA or easer way is apply these with the MCU(microcontroller unit) or using computer-on-module. There are lots of companies provide the computer-on-module hardware solution for the different specific scenario. Such as NVidia jetson COMs, The Google Coral AI, Raspberry Pi compute, The OpenMV. They are different prices range and the different computation ability for different scenario. For example the Nvidia jetson is platform is a highly specialize for running embedded TensorFlow models and have the excessive power for numeric datasets, it can use for the image video processing application, Tesla auto driver is early develop using this platform. The Raspberry Pi is the excellent for prototyping small compute project, it can easily scale to production. The OpenMV is a similar platform for computer vision in small computer project. However the OpenMV ready-use-system the long term availability is doubtful, this product is lunch in April 2018 but it is no longer available after july 2019. The life cycle is too short. The system for OpenMV is no longer update. Instead the Raspberry Pi the open source characteristic is good for long term use and it have a good expandable ability. In this project is developing the guidance system platform as the upper compute unit. The system aim to have the low cost easy attachable for the lower execute unit. The Raspberry Pi will be suitable for the low cost and long tern availability for develop the project (Ettlin & Bleuler,2006; Kroeger, Huegle & Niebuhr, 2019).

Aim

The final product of this project is to build an All-Terrain robot platform use in the ELEC2209 course taught. More specifically, this project focuses on the master computer, or the guidance system, of the developed robot. The aim is therefore to develop, evaluate and make a very low-cost computer vision system for the robot openly available to students in the course. To relax the complexity of system implementation, a micro-controller using Raspberry Pi will be developed to control a simple robot. With the guidance system implemented, the robot is expected to adapt and follow a visually marked out trail under various physical environments. By the end of the project, an easy and reliable vision system will be built and validated for the robot. The developed vision system could be further applied to the hardware aspect of the robot to adapt outdoor all-terrain trail.

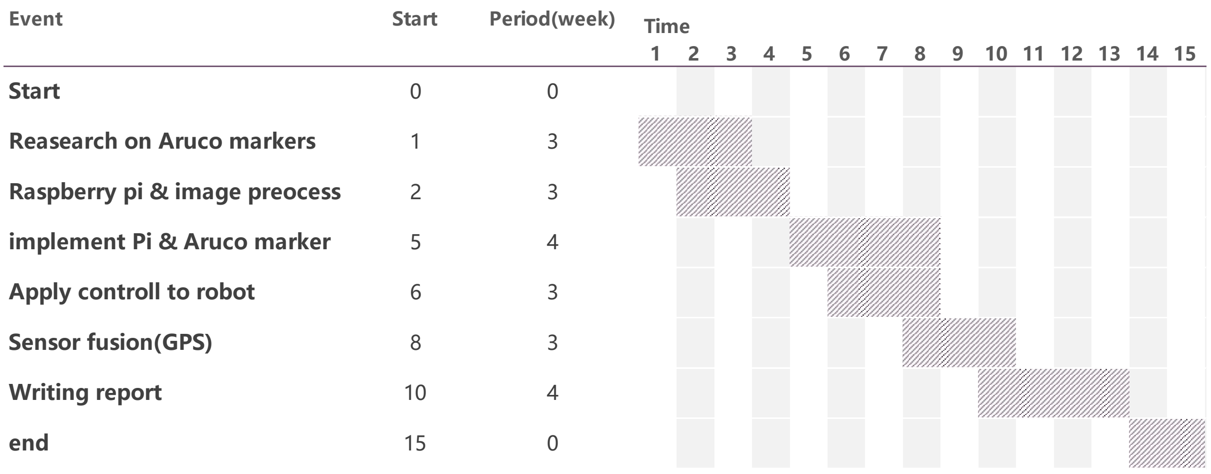
Background & Objective

In order to navigate through its environment a robot vehicle needs to determine its position relative to certain objects or landmarks. If the robot is going to follow a pre-defined path then the information it obtains must provide its position relative to the path. If the path is not pre-defined then the robot needs to know its position relative to the obstacles which it must navigate around. In both situations a flexible and quick method for relative positioning is necessary

The guidance have passive and active tern. The guidance is passive if no localization of guided objects is place(the guiding sign the marker, lighthouse,etc) and is active when it involves localization and have some kind of communication with the guided object. This project will use raspberry pi zero and the camera combo and use aruco marker as the active guidance. Guidance system use in all-terrain robot usually consist two main part the positing and correct. Use cameras, sensors to collect the environment character such as using the camera fusion with IR, ultrasonic sensor to build up computer vision and using gyroscope to analysis the current gesture. Then the processing unit to calculate the information form different sensors. The use of Aruco code can be use in the camera to calibrate and positing the position. The IR and ultrasonic can be use as collision sensor detecting on other side of mobile robot. The gyroscope can collect have the contain data preprocess by filter the noise etc… and get the data such as the facing angle, the movement angle speed and the 3D Acc value by integral the data to get the gesture facing and movement speed. The better algorithm will enforcement the ability of the robot. After the position and correcting data been generate will output to actuator to execute the correction. During the correction will also have algorithm known as PID controller ADRC controller fuzzy controller and even the simplest pangpng controller, each of them one has different pro and cons base on different environment and scenario. The project hardware will use base on Raspberry pi zero and the Arduino base controller such as Ardu-pilot or F7 fight controller which can use GPS signal to do the advance positing. The robot design to use in all-terrain environment with the image process and senor fusion as the guidance system.

2.Project management

Due to the COVID-19 impact the time manage and the situation for project is unknown. For the project actually dose is different quieted different from the Gantt chart. However, The object is mostly archive and satisfy the aim discus in *result*. Due to the autonomous robot is not available at the time stay in the living place. the guidance system is tested on pure serial output result, can see *test and validation.*



Event	Start	Period(week)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Start	0	0															
Reasearch on Aruco markers	1	3	▨	▨	▨												
Raspberry pi & image preocess	2	3		▨	▨	▨											
implement Pi & Aruco marker	5	4					▨	▨	▨	▨							
Apply controll to robot	6	3						▨	▨	▨							
Sensor fusion(GPS)	8	3								▨	▨	▨					
Writing report	10	4										▨	▨	▨	▨		
end	15	0														▨	▨

Figure_2: the Gantt chart for project management.

3.Relate work

The flowing literature review is the preparing technical content for technical approach.

Fiducial markers

The general fiducial marker is like the square shape with a pattern image inside which contain the encode information. Using square shape is popular due to the four special point that is easy to detection. Which allow camera to calibration and marker position calculation. In order to increase the detection sensitivity most of marker system is use monochrome marker. The monochrome marker is only in black and white color in the marker. This way of detection can avoid the grey scale instead with the threshold decision only to per pixel. (Garrido-Jurado et al., 2014)

Aruco Markers

ArUco is a popular method for vison-based fiducial marker. It was an AR kind high reliable marker, originally developed by Rafael Muñoz and Sergio Garrido to generate set of AR fiducials and to simplify the estimation of 6DoF marker poses.(Figure_3.1) The Aruco marker is based on C++'s OpenCV. It can be used to detect of different tag direction and can be using to calibrate the positing or be as the sign as some special usage. The pro of this Aruco marker detection pipeline is much more effective and faster than the similar AR fiducial marker AprilTag.

The Aruuco markers are synthetic square markers with outline a wide black border with an inner binary matrix contain the information. The black border can be using the fast detection and position of the image and binary matrix could be marker identification. The encode of binary matrix Aruco allow to error detection and correction technique (Sagitov et al., 2017).

The detection process is performed in two main steps, the detection of marker candidate and codification analysis. In the recognition of marker, will set an adaptive thresholding(see section *detact marker*) and contour extraction with the extract filtering in order to find out the shape of square. This is the candidate to be markers. Then is to analyze the inner encode pattern. After the candidate detection, we need to check the detected marker are actually markers by analyzing their inner codification. By extracting the binary bit of each marker. To decode the image, we need transform the marker in to the regular square and divided in to different cell reference by marker size. Set the threshold to determine the black cell bit or the white cell bit. Finally, the bit are analyzed to determine if the marker are the specific dictionary.

Figure_3.1. The example of Aruco marker with different bit size

Pose estimation and mapping

The camera we can imaging with the world origin. The marker position is calculated in respect to the camera capture frame. The relationship with the M-marker and the C-camera frame can express as the marker matrix can been transformation by T_{mc}

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_c = T_{MC} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_m \qquad (1)$$

The T_m can calculate as:

$$T_{\mathrm{MC}} = \begin{bmatrix} (e_0^2 + e_1^2 - e_2^2 - e_3^2) & 2(e_1e_2 + e_0e_3) & 2(e_1e_3 - e_0e_2) & t_X \\ 2(e_1e_2 - e_0e_3) & (e_0^2 - e_1^2 + e_2^2 - e_3^2) & 2(e_2e_3 + e_0e_1) & t_Y \\ 2(e_1e_3 + e_0e_2) & 2(e_2e_3 - e_0e_1) & (e_0^2 - e_1^2 - e_2^2 + e_3^2) & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2)$$

The e_0, e_1, e_2, e_3 is the quaternion components and t_x, t_y, t_z is the translation of marker position. Use this method allows us to estimate marker pose whether the camera is identified as world origin, but for mapping purposes an inverted approach in which the camera position is estimated with respect to the static markers is more convenient.

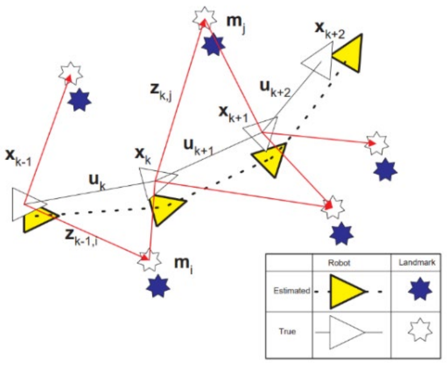


Figure_3.2:. estimation and mapping

PID controller

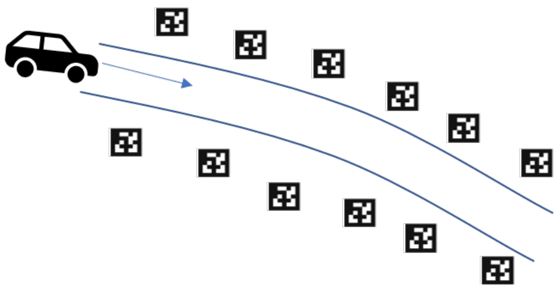


Figure_3.3:. PID apply scenario

The above figure_3.3 is the testing scenario by recognize the marker to keep robot in the track. We will introduce PID controller to control the car.

PID full name is proportional- integral -derivative controller. It is a control loop feedback mechanism. Widely apply in our daily life and industrial used. Such as the car speed control, the motor control. PID controller calculate the error value continuously as the difference between desired target. And correct on the proportional, integral and derivative terms. The overall control function can be expressed mathematically as:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{3}$$

Where K_p, K_i, K_d, are not negative number donate the coefficient for the proportional, integral and derivative terms. PID controller use in this project it mainly to control the speed of two side wheels to keep the mobile robot in track. The control block is the PID controller control diagram. w

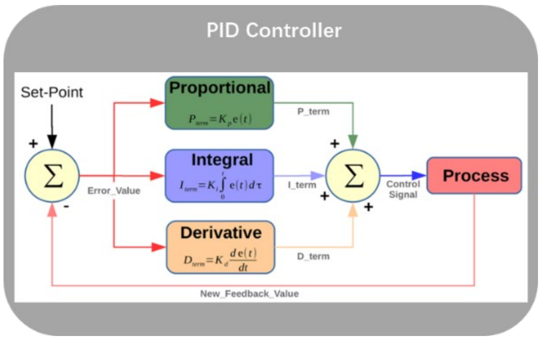


Figure_3.4: the PID controller control diagram.

4.Technical approach

4.1 Robot Ecosystem

The Autonomous guidance system platform hardware is built on raspberry pi zero. It is opensource microcomputer platform, The Raspberry Pi OS is based on Linux system using the bash command and built in python as the development tool. Raspberry pi gives a light way, low cost computation platform for light computational project, which is highly used in robot, embedded project, and Internet of things project. The software is developed under Python version of OpenCV, which is an open source computer vision framework. OpenCv is essentially a C++ API developed. OpenCV core function consists of the image processing, video analysis, the object detection. In this project, the code uses marker detection module to process the computer vision. The OpenCv provide the video I/O easy use interface to video capturing and video codecs. We are using the Pi camera as the hardware input I/O. The basic array computation and the math operation is using Numpy library. Numpy provides a light high compute efficiency array operation. It is necessary to apply highly efficient computation package to produce real-time computational task.

In the whole guidance system, the raspberry pi is considered as the upper computer provide the image process result and sent to the lower compute execute unit. We use the UART protocol and GPIO output to communicate the lower execute unit. Detail see *Serial Terminal Ouput* and *Hardware I/O mapping*. This project only developed the upper computer as the service provider. Lower compute can be implemented as the microcontroller such as pic microcontroller, the Arduino microcontroller, the FPGA chips etc…

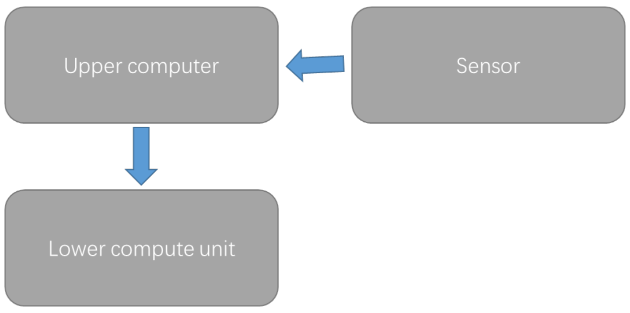
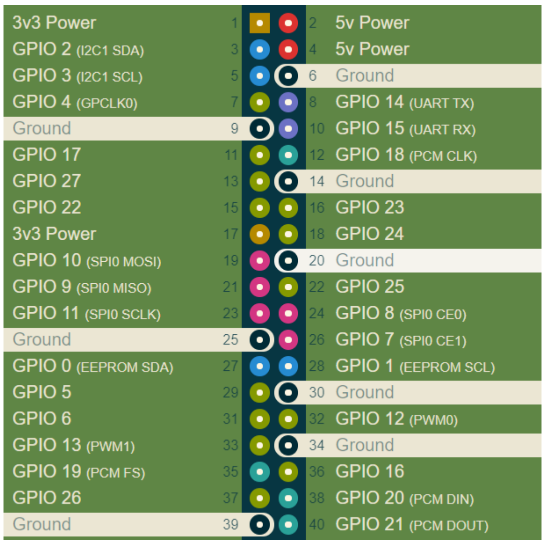


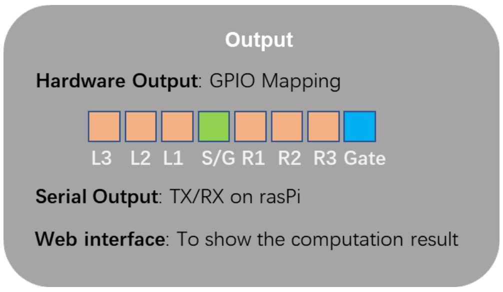
Figure_ 4.1: upper service provider and lower compute unit

Interface

The guidance system upper computer raspberry pi mainly has 3 outputs for lower computation unit. The main output is 8 GPIO pin on raspberry pi with the image process result. Each pin outputs a logic high or logic low 3.3V/0V Boolean signal to represent True or False of each config pin. Addition information is configured by the UART Serial communication pin ***GPIO14,GPIO15*** TX/RX. Detail see *Serial Terminal Ouput* and *Hardware I/O mapping*. Also, the image process preview from camera are assigned one special I/O config pin to be triggered on the AP, allowing user to visit the host website. This web preview is based on the Flask WebSocket framework. The purpures for setting a web camera view is to help user to install the camera and put the camara in a correct position. Additionally, the web camera monitors the image process result to check if the GPIO output set is configure correctly.



3v3 Power	1	2	5v Power
GPIO 2 (I2C1 SDA)	3	4	5v Power
GPIO 3 (I2C1 SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27	28	GPIO 1 (EEPROM SCL)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)



Output

Hardware Output: GPIO Mapping

Serial Output: TX/RX on rasPi

Web interface: To show the computation result

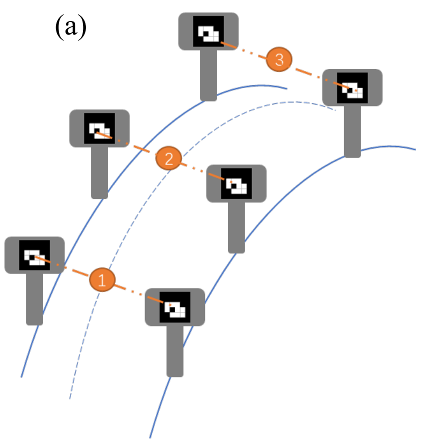
(figure_4.2)

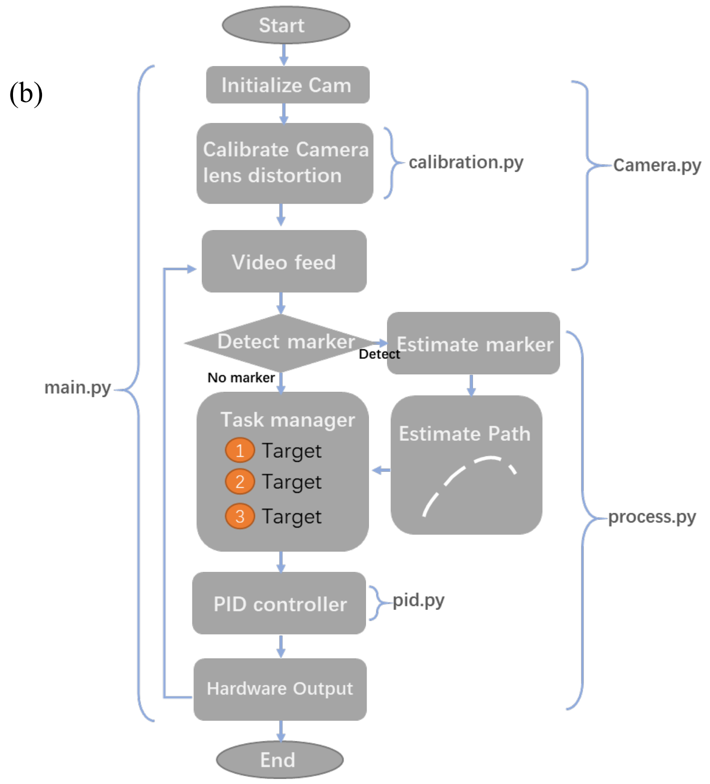
Pin config	L3	L2	L1	Stop/Go	R1	R2	R3	Gate
GPIO	17	27	22	5	23	24	25	6
BCM	11	13	15	29	16	18	22	31

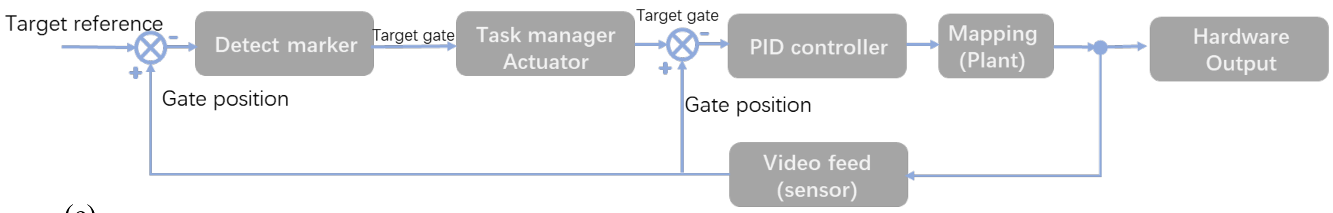
L3 - L1 and R1 - R3: Represent the degree of deviation from the target marker, each epoch only encodes on bit of pin.
Stop/Go: Represent the status should be stop or go, if is false then is output logic low 0V
Gate: Represent if there is any gate target to follow. If there is above 1 gates detect will trigger as True 3.3v logic High

4.2 Software Architecture

Figure_4.3 (a) and figure_4.3 (b) illustrate the software workflow and the path demonstrate example. Whole program has four class to complete the project aim. There are ***`calibration.py`***, ***`camera.py`***, ***`process.py`*** and ***`pid.py`***. The whole guidance system is based on camera video feed as positive feedback loop see figure_4.3 (c), which execute result is feedback on each epoch. Before each epoch, the camera calibration is necessary to find the distortion matrix to fix the camera lens distortion before use the image to detect markers. The ***`calibration.py`*** is serve the main calibration process. The camera calibration detail can see section *camera calibration.* The input of the program is the video stream from Pi camera, which is set up by ***`camera.py`***. The epoch of the program is to process the video stream from every frame. The main image process is compiled in the ***`process.py`***. The first step in the epoch is to apply the lens distortion make sure the frame from video feed has been correct. Then detect the marker in the frame. It there is marker then then the marker fit the gate condition the program will estimate the path and generate the target to the task manager. If there is no marker detect or other issue cause the marker detection failure will directly run the task manager to deiced the action. The failure case can reference from *Test and Validation*. The task manager is design as a actuator to directly control the robot action, manager will decide what to do next and have some prediction for the executing task such as the time estimation to reach target or failure protection etc. The detail of Task manager can view *Task manager* The output of the task manger is the target error. The error will feed to PID controller in ***`pid.py`***. then the output of PID controller will be mapping in to the bit wise hardware output. The hardware output configure detail can see section *Interface* and the detail of Hardware implementation see *Serial Terminal Ouput* and *Hardware I/O mapping*. From the block diagram view figure_4.3 (c) the Detect marker and Task manager is the higher level of the Actuator and the PID controller is the lower Actuator the output from the PID controller(see section *PID controller*) will be mapping into the bit wise hardware output.







(c)

Figure_4.3: (a) The demonstration of environment left part. (b) Workflow guidance system: initialize Pi camera, Lens distortion fixing, marker detac and estimation, task manager, the PID controller and hardware output. (c) The block diagram of guidance system

Aruco marker generation

Before we setup for the robot, we need select the type of Aruco dictionary first and generate Aruco marker to setup the environment. The Aruco module provide the different format of Aruco marker to gives different use in the application. There are consist of different dictionary size of number in the dictionary and the marker size, the number of bits in the marker. Below table is the dictionary set of Aruco marker. For example: DICT_4X4_50 there are two part first 4X4 is the marker bit width of the marker from figure_4.4 there are 4X4 and 6X6 markers, the more bit can be, the more information and more tolerance can be. Second part is the number 50. This illustrate the dictionary size, 50 means there are 50 different markers in this dictionary.

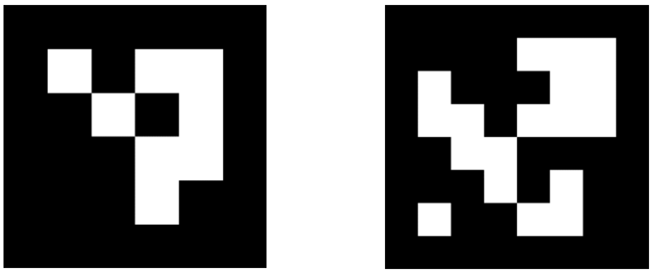


Figure_4.4: Aruco marker 4X4_1 and Aruco marker 6X6_1

In this project we choose 4X4_50 as the gate marker. 4X4 is the least bit on the marker dictionary (See *Appendix B*), the less bit mean on the same area of marker each bit have the larger area to represent one bit, It can be optimize the fiducial marker detection success rate in the low light or in the lower resolution. The less bit mean faster speed. In this project the gate marker only needs to hold the basic information the gate id.

Camera Initialize and Calibration

Initialize Pi cam

From the *library*, ***picamera*** lib provide the I/O interface to access the Pi camera hardware. In the project is compact into ***camera*** class to initialize and provide the video capturing in a easy way. Just provide the resolution needs to capture from Pi camera. In the project the default resolution is 640*480 and 30 FPS.

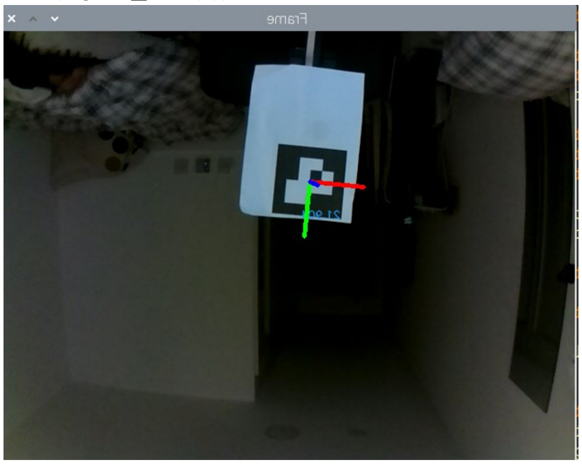
```
    #initail raspberry pi camera
    p_camera = calibration.camera(w=w,h=h,framerate=frame_rate)
```

The camera now is initializing and ready to use. For the choosing of resolution can see the *test and validation* discuss about the resolution tat affect the whole progress.

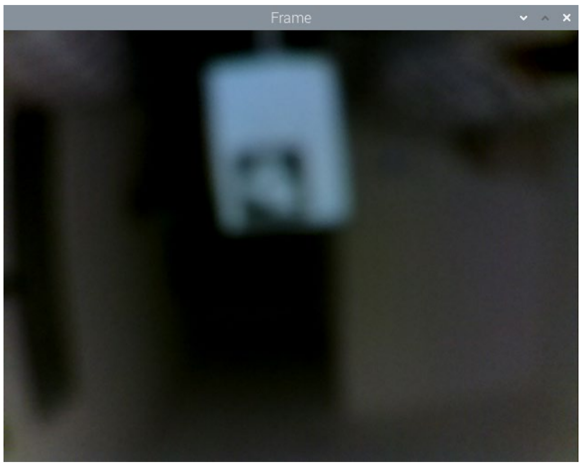
Calibration

To make sure the Pi camera output image is correct and useable. The camera calibration is necessary step before the image processing to make the better result. There are two main step to calibrate camera one is adjusting the focus length, then obtain the distort coefficient to fix the lens distortion. The first step is make sure the camera is in focus, then we will use the ***calibrate_camera()*** compact in `calibration` class to fix the lens distortion

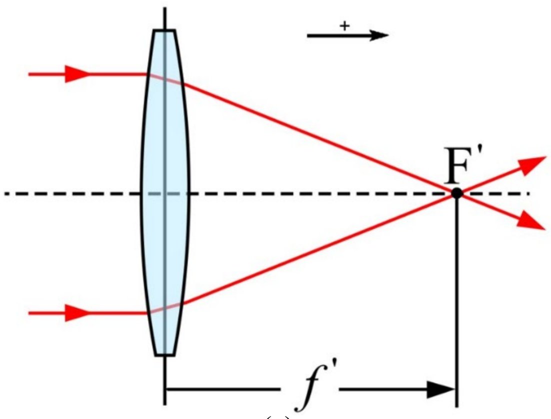
1. The each Pi camera from may have set in different focal length in default factory status. From the figure_4.5(a)(B) We can see that the difference of the camera lens is in focus and out of focus. There are due to the focal length is too close or maybe it is focal length is too far. The focus lens length is fixing to change the in/out of focus is to adjust the pi camera lens by spin clockwise or counterclockwise the case of the camera until the image is clear enough for detecting the marker. (figure_4.5 (c))



(a)



(b)



(c)

Figure_4.5: (a)The camera lens focus point is focus on the sensor. (b) The camera lens focus point is not lay on the sensor, which is out of focus. (c) The relationship of the camera lens and focus lens.

2. The software Camera calibration is consists in obtaining the camera intrinsic parameters and distortion coefficients. This parameters remain fixed unless the camera optic is modified, the camera calibration only need to be done once at the very begin of setup.
   a) In the **`photo`** folder see *project structure* there are pre take different viewpoints of ChArUco, since the pi camera is using the same lens, all the pi camera can use the same pre take photo. The example of the pre take is like the following Figure_4.6

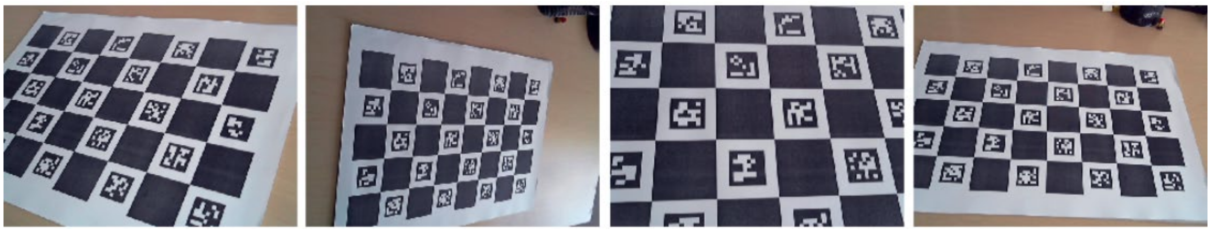


Figure_4.6 : Different view point of ChArUco in **`photo`** folder, see *project structure*.

   b) For calibration, the camera is supper essay to implement in the project as the flowing line. After initializing the pi camera. The image capture from pi camera is useable for the post image process

```
  image_process = calibration.process()
```

Detect marker

The marker detection is using the ArUco module in the OpenCv library. By giving an image containing Aruco markers, the *detectMarkers()* will return the detect marker corners and the *ID* of marker. The image capture from Pi camera is 8bit RGB color space, this format is the human eye can see in the real world. However the computer vision only can read number which is the gray scale [0,255] of R, G, B channels. In order to detect marker more sensitive and sense the all characteristic of RGB in three channels instead of reading one channel of data. It is necessary to transform the the image Capture from Pi camera video stream feed from RGB/BGR color space transform into the grey scale. The figure_4.7 Shows the colors project in the RGB color space. To make the full information is project in greyscale the formula is following.

$$RGB[A] to\ Gray: Y \leftarrow 0.299.R + 0.587.G + 0.114.B \quad (?)$$

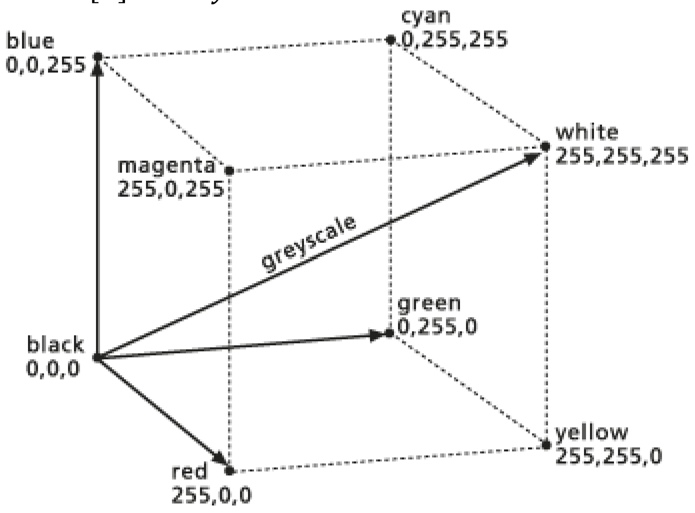


Figure_4.7 The color information project in RGB color space and the relation with the grey scale.

After the color transformation. The parameter *adaptiveThreshwinSizeMax* to set the auto exposure to detect the marker, it may necessary when is to optimize the marker detection. The following table is the list of parameter it can adjust for optimize the detecting markers. The *corners* and *Ids* will be return from *detectMarkers()*. These two variables is will be use to localize the marker latter. The result will pass to the path panning algorithm.

adaptiveThresWinsizeMax
adaptiveThresWinsizeMin
adaptiveThresWinsizeStep
cornerRefinementMethod
cornerRefinementSize
cornerRefinementMinAccuary

Table1: the list of parameter can be adjust to optimize the accuracy.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_4X4_50)
parameters =  aruco.DetectorParameters_create()
parameters.adaptiveThreshWinSizeMax =23
corners, ids = aruco.detectMarkers(gray, aruco_dict, parameters= parameters)
 # SUB PIXEL DETECTION improve cornor accurate binning effect
corners_corection =None
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.0001)
    for corner in self.corners:
        cv2.cornerSubPix(gray, corner, winSize = (3,3),
        zeroZone = (-1,1), criteria = criteria)
```

The successfulness of the detecting marker *detectMarkers()* is affected by the camera resolution and also many other issues, to optimize the *detectMarkers()* results can adjust the parameter. It will discuss this issue in *test and validation*. The sub pixel detection can be use to improve corner accurate, see the *binning effect*. The *cornersubPix()* is to fixing the binning problem

Euclidean Space localization

The image capture from pi camera is the camera perspective in 2D, to calculate and estimate the pose in the 3d world. We need to finds out the relationship between world coordinate system and camera coordinate system by using the equation in *pose estimate and mapping*. In the OpenCv they provide *solvePnP()* to finds an object from 3D 2D correspondences. Function estimate the input marker object point that project on image corresponding, see the figure_4.8 The marker coordinate system is place at the center of the marker with the depth is represent Z-Aix the X and Y is represent the frame X and Y. The function *estimatePoseSingleMarkers()* is called *solvePnP()* transfer the coordinate system and returns the markers rotation and translation vector. In the Project we only need to knows the translation vector to get the distance to the camera. After getting the distance to the camera we can start the path planning and figure out where to move.

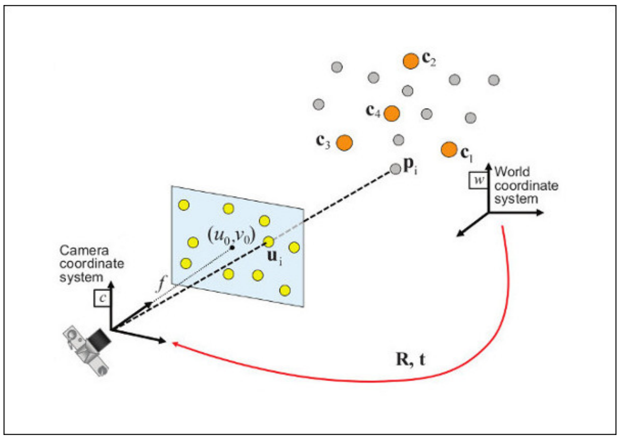


Figure_4.8:X- aixs of camera is point to right, the Y- -aix is point up and down, and the Z-aixs is the depth of the marker.

Path planning Algorithm

It is necessary to locate the detecting marker in the list when detect and transpose location process is finish (Khan et al., 2008). The order obtain in each epoch is different the bellow pseudo code shows how to sort and calculate each transpose location of each marker in the image and finds the Gate amounts the markers.

```
Algorithm: Locate gate marker index and sort gate in order in each epoch
1. Get ID, Corners of marker <= detectMarkers()
2. Get The rotation of marker, transpose of markers <= estimatePoseSingleMarkers()
3. for n = 1: Number markers do
        Distance_list[n] = getDistance (transpose of markers[n])
   end for
4. sort_dis = Sort(Distance_list)
5. sort_index = empty
6. #The Sort_index store each markers rank in the detacting list
   for n = 1: length(transpose of markers) do
        for m = 1: length(transpose of markers) do
             If Distance_list[n] equal sort_dis[m] do
                  Sort_index[n]= m
   end for
7. pos1, pos2, pos3, pos4, pos5, pos6, Index = 0
8. #Place the order index of marker in the pos
   for id in sort_index do
        Index++
        If the id equal 0~6
        pos[1~6] = Index
   end for
9. gate_index = [pos1, pos2, pos3, pos4, pos5, pos6]
10. Reshape gate_index as [ pos1, pos2] #gate1 the closest gate
                          [ pos3, pos4] #gate2 the 2nd closest gate
                          [ pos5, pos6] #gate3 the 3th closest gate
11. Return gate_index
```

Above we locate the gate information in the detecting list. The following pseudo code illustrate the path generation and extrapolate the path make the path more accurate.

```
Algorithm: Generate Path form gate index
1. Get Gate_index and locate the gate markers in transpose list
2. Path_list <- caculate the midpoint of each gate in transpose list[Gate_index]
3. Upsample Path <- extrapolate Path_list up sample to smooth the path
4. Return Upsample Path
```

To Validate this feature can be seen in *test and validation*.

Task manager

The program use the task manager to control the robot. Task manager is design to manage all the information coming from each module. It have the function to prevent the detection failure or losing objective due to some of issue make the robot stop in the flowing path. The detail of cause failure is discuss in *Test and validation* or Analysis section. In task manager by detecting each frame of markers distance and record the time of each frame to estimate the **speed of robot** and estimate the **arrive time** to the gate. If failure happen task manager can handle and continue execute the previous task to approach the closest gate and waited to find the next gate. The output of Task manager is the *target gate* as seen in the figure_4.3(c). Figure_4.9 is showing the ideal test case. In ideal situation task manager will deliver the target gate pass the location information to PID controller to finish the path, during the execution the gate has pass out the camera view but the location of robot is not approach the gate location. The task manager will using the last compute result to continue execute to approach target by reference **speed of robot** and **arrive time**. the bellow figure shows the camera view of this situation when robot is passing by the gate. The decision making for this situation can see the *test and validation section*, to test the failure feature of this algorithm.

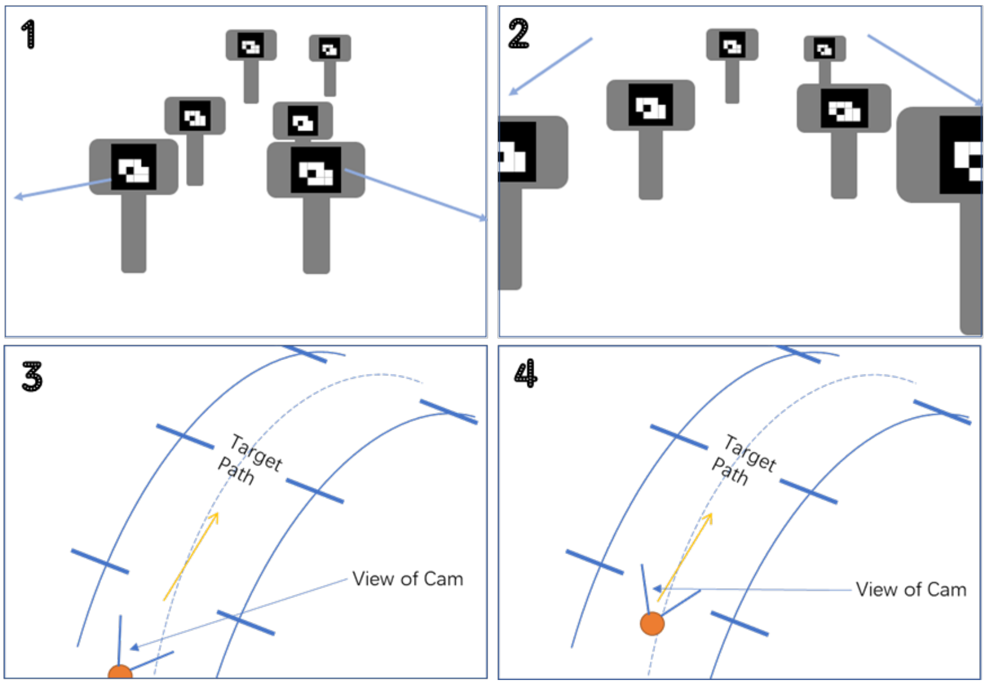


Figure_ 4.9: (1)(3) Robot to closing to first gate (2)(4) Robot is passing the gate. (1) and (2) is the camera view point. (3) and (4) is the top point of view sowing the Robot scene

Hardware I/O mapping

From the figure_4.3 controller block diagram. The output of PID controller will pass through the plant to mapping and limit the output of PID controller. Then finally sent the signal to the hardware output. The GPIO pin configure can see *Interface*. Flowing is using Arduino analog pin to test the output result on each GPIO configure pin. The Arduino hardware test file can be found in ***Arduino testset*** in *program structure*. The figure_4.10 is shows the Gate I/O pin test output. The raspberry pi logic high output is 3.3v and Arduino analog input range is 0~5v 8bit. The value read in Arduino Terminal is 675 when the output is HIGH we can calculate that $\frac{675}{1024} \times 5 = 3.296v$. From the reading result the output signal is correct.

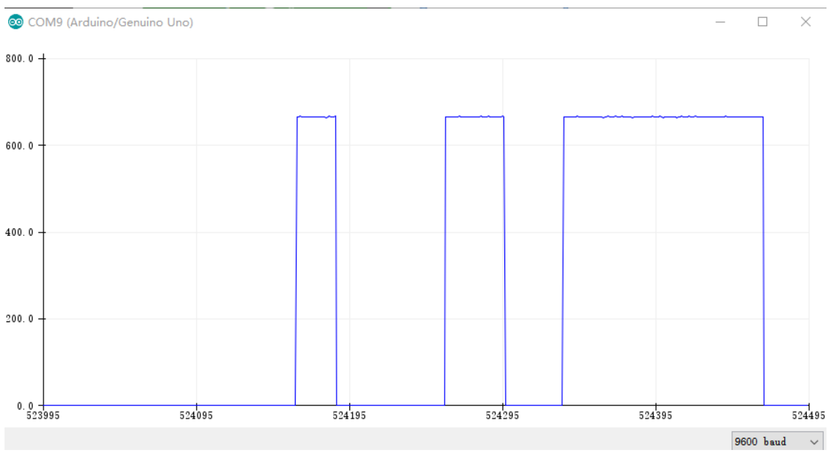
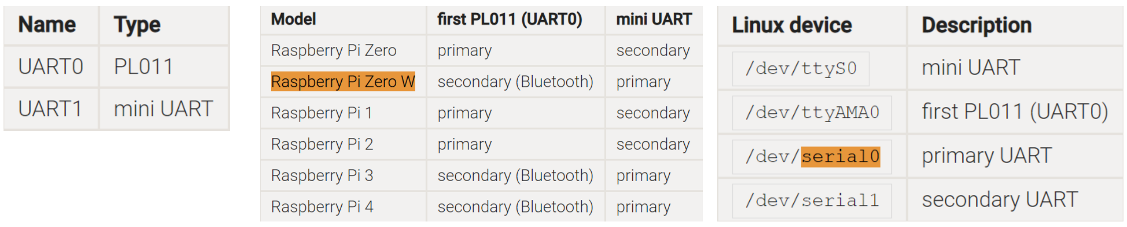


Figure_4.10:Arduino analog pin receive Raspberry pi I/O Gate pin plot graph.

Seral terminal output

Serial communication is a efficiency way to transfer data amount the device. The data will be sent parallel in the same time. In raspberry pi have UART protocol to transfer data. UART stands for “Universal Asynchronous Reception and Transmission”. UART is a Multi-Master protocol. The connection between two device can send data free when they want. This protocol is difference from other communication protocol such as I2C or SPI Master-Slaves configuration. The communication will be more freely to transfer data amount each device.

From Figure_4.11 There are two UART terminal port on the pi zero UART0 and UART1, and there are two GPIO pins: ***GPIO14,GPIO15*** are default configure as the UART protocol see from *4.1.2 Interface*. By default the UART0 is enabled as the primary UART. The port device of UART in Raspberry Pi OS is ***serial0***.



Name	Type
UART0	PL011
UART1	mini UART

(a)

Model	first PL011 (UART0)	mini UART
Raspberry Pi Zero	primary	secondary
Raspberry Pi Zero W	secondary (Bluetooth)	primary
Raspberry Pi 1	primary	secondary
Raspberry Pi 2	primary	secondary
Raspberry Pi 3	secondary (Bluetooth)	primary
Raspberry Pi 4	secondary (Bluetooth)	primary

(b)

Linux device	Description
/dev/ttyS0	mini UART
/dev/ttyAMA0	first PL011 (UART0)
/dev/serial0	primary UART
/dev/serial1	secondary UART

(c)

Figure_4.11: (a) The raspberry pi zero UART port. (b) The configuration for UART. (c) The UART Linux devices on Raspberry Pi OS. https://raspberry-projects.com/pi/programming-in-c/uart-serial-port/using-the-uart

The following Bash command will set read and write access permissions for all users on the UART.

```
sudo chmod a+rw /dev/serial0
```

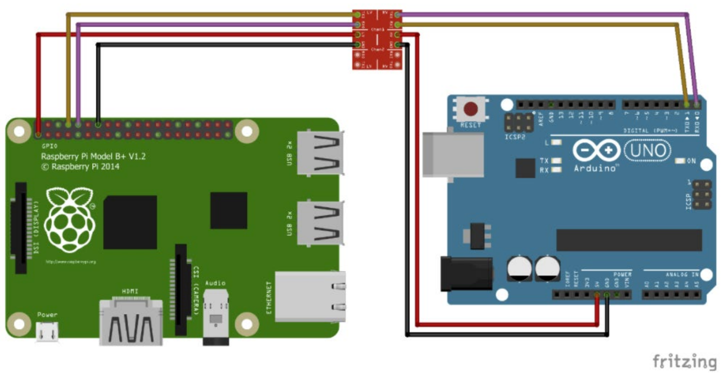
The following lines are set for open the Serial communication in the beginning

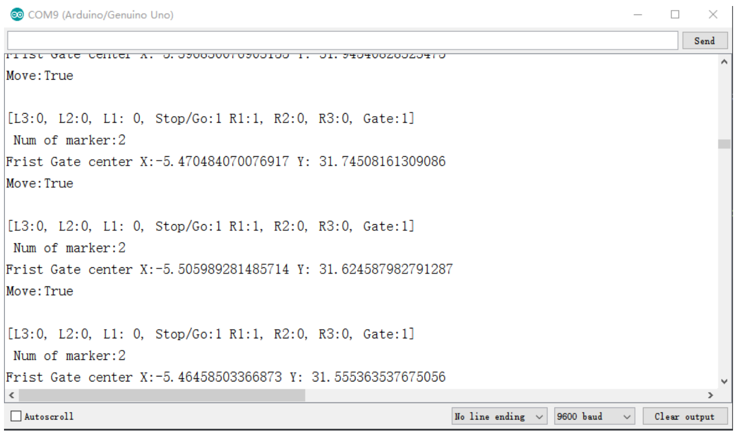
```
import serial
serial = serial.Serial('/dev/serial0', 9600, timeout=1)
ser.flush()
```

After the *update()* the image process result.

The following pseudo code demonstrates message packing. The final output of serial terminal output will be send by *Serial0* port in hardware TX/RX **`GPIO14,GPIO15`**.

```
Algorithm: UART Send Message
1. Initialize Serial0 port in band_rate =9600
2. Get hardware I/O result from I/O mapping
3. String1 = I/O mapping result
   String2 = get Num of Marker
   String3 = The closest Gate in Euclidean Space location in camera point of view
   String_Out = empty
4. If Marker_detection() is True:
         String_Out = String1 + String2
   If  Gate is available and  Marker_detection() is True:
         String_Out = String1 + String2 +String3
   else: String_Out = String1
5. Return decode( String_Out ) in Byte the format of 'utf-8'
6. Write Return string in Serial0 port
```





```
Move:True

[L3:0, L2:0, L1: 0, Stop/Go:1 R1:1, R2:0, R3:0, Gate:1]
 Num of marker:2
Frist Gate center X:-5.470484070076917 Y: 31.74508161309086
Move:True

[L3:0, L2:0, L1: 0, Stop/Go:1 R1:1, R2:0, R3:0, Gate:1]
 Num of marker:2
Frist Gate center X:-5.505989281485714 Y: 31.624587982791287
Move:True

[L3:0, L2:0, L1: 0, Stop/Go:1 R1:1, R2:0, R3:0, Gate:1]
 Num of marker:2
Frist Gate center X:-5.46458503366873 Y: 31.555363537675056
```

Figure_4.12 : (a) connection with Arduino. (b) The Terminal output using Arudino IDE, receive the message from Raspberry Pi

Figure_4.12(b) is the example of output result by using Arduino serial console. To read the result you can use any serial port reading software such as PuTTy, the Serial port Utility etc… Figure_4.12 (a) is the connection using Arudino and the Raspberry Pi. The arudino set up can reference in *project structure*

5. Software Metadata

Platform

The code was developed using Python version 3.7.3 on the open-source platform Visual studio code, validate in Linux Raspi OS (Desktop version). The hardware use is Raspberry Pi Zero W. The testing device is using Arduino Uno and the Arduino IDE compile the test program.

The link to the developed code: https://github.com/Joseph-tsai415/Msc-All-Terrain-Robot

Libraries

The following libraries are utilized for code development.

1. NumPy (v 1.19.1)
   NumPy consists of various numerical computing routines for multidimensional array.
2. SciPy (v 1.5.2)
   SciPy is a free Python fundamental library for scientific computing. It offers user-friendly tools for numerical computation.
3. picamera (v 1.13)
   Picamera is provide the camera I/O interface in Raspberry Pi OS, it offers user friendly access the camera device in Raspberry Pi system.
4. RPi.GPIO(v 0.7.0)
   RPi.GPIO is use for hardware GPIO
5. Opencv (v 4.1.1)
   OpenCV is an open source computer vision and machine learning software library*(OpenCV team, 2020)*. © Copyright 2020, OpenCV team
6. Pyserial (v 3.4)
   Use for the UART connection

Project Structure

The whole software is packet into the python format library. The project structure for the project is following.

```
├── Project structure tree
│ ├──Calibration
│ │├─── marker_gen
│ │└─── marker_gen.py
│ │├─── photo
│ │└─── take_pic.py
│ ├── __init__.py
│ ├── calibrate.py
│ ├── camera.py
│ ├── pid.py
│ ├── process.py
│ └── Readme
│ ├──arudino_testset
│ └── arudino_testset.ino
├ main.py
└ autorun.sh
```

`Main.py`: This is the *main()* program, this is the start point of whole program.

`__init__.py`: Use to package every module into packages.
`Calibrate.py`: this file is compact of the function: camera calibration. This function is fixing the distortion problem happen in the camera lens and provide the correct fix distortion matrix to fix the image. The file is implementing in a class name calibrate. This class will called photo folder to execute calibration.

The ***`photo`*** folder: It is content the default pre take picture of different angle of the ChArUco marker board use for calibration.
The ***`marker_gen`*** folder: Use to generate different kind of marker for calibration or the path marker use.

The **`Arduino_testset`** folder: This folder contain the Arduino test set use to validate the Raspberry Pi hardware I/O and UART communication working correctly also the provide user the addition information test platform in Arduino.

`Camera.py`: This file compact of the raspberry pi camera initialize setting the file is implementing into class name camera. It provide the video resolution setting, the frame rate setting, and the video stream feed function.

`Pid.py`: This file is providing the PID controller use for path following and the task executor. The main function of this file is PID controller and it is package into PID class.

`Process.py`: This file is called process.py, it contains the class: *process()*, this is the main function of the program, it provide the image process of whole service during the run time. It contain the *update()* function will detecting the marker and estimate maker position, then find the gate and predict the path. This function needs the video feed in from the camera class. It also contains the I/O mapping to mapping the computation result into the hardware output the hardware output is mange by the task manager. The task manager function is also in this class, it will manage the detecting gate in the update function, predict the path and estimate the time arrive to avoid the detection failure. To arrange the task for the robot to execute on the I/O mapping.

`Autorun.sh`: This shell script is the launcher of whole program. Before execute the python script on the robot need to create the launcher to prepare the system. Access the python virtual environment and then executes the python script. The following code is she content of autorun.sh for the guidance system for start up.

```
#!/bin/bash
. /home/pi/.profile
. /home/pi/.bashrc
workon cv
sudo chmod a+rw /dev/serial0
/home/pi/.virtualenvs/cv/bin/python3 main.py
```

User environment setup guide:

1. For setting of the environment, we firstly need to install the Ras pi OS(Desktop version)and the OpenCV can follow this link as reference to set up. For Raspberry Pi Zero w the install time will depends on the compute ability and the internet connecting speed. For my cause I speed almost 30 hours to complete the installation of the OS and OpenCV.
   The link: https://www.pyimagesearch.com/2019/09/16/install-opencv-4-on-raspberry-pi-4-and-raspbian-buster/
2. Install the python lib in the lib list
3. Setup the AutoStart run by the link
   The link: https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/

6.Test & validation

The test and validation is important to testing the robot system robustly and reliability. In this section, we will try to find the limitation in some factors: the ambient light, the maximum angle of marker can detect, the function achievement and the failure reaction test. Due to the effect of Covid-19 the test environment is limited in the living bedroom condition. It can be developed and tested in more different scenarios in the future. The following has few experiment to test different feature I mention in precious discuss.

1. **Experiment: The influence of Ambient light to the marker detection:**
   The bedroom light setting: The following figure_6.1 is my rooms settings, there is a 400lumens bed

lamp on the table, a 1600 lumens stand light beside the table, and a 2000 lumens room light at top of the table, and a 2000 lumens room light at back of the room. The light will conbine in different combination to test the Guidance system robust ability. The camera is place on the table, the *testing set* face 90degree and place 25cm away from camera. The way to measure how good is just see how many marker can be detected in each test case.

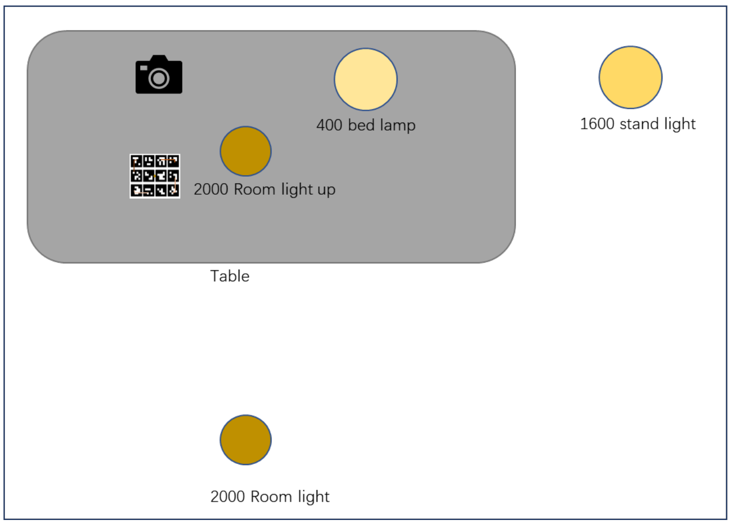


Figure_6.1: the top view of my bedroom light setting, different position is will affect environment differently.

In order to challenge ability of the detect ability also the parameter we set in the *Detect marker setion*, we will set different light environment, which has the backlighting condition and the frontlighting condition also in different angle. The fowling is the different combination of lighting.

a) 1600 + 2000 up+2000 back:
   Turn on the 1600 standing light on the side of table, turn on the room light on my table, and the back of room light.
b) 1600+2000 up lumens:
   Turn on the 1600 standing light on the side of table, turn on the room light on my table1600 lumens. Turn off the rest of the light.
c) 400 lumens extreme low light test:
   Just turn on the 400 lumens bed lamp:
d) 2000 back lumens extreme strong backlighting:
   Only turn on the back of room light.

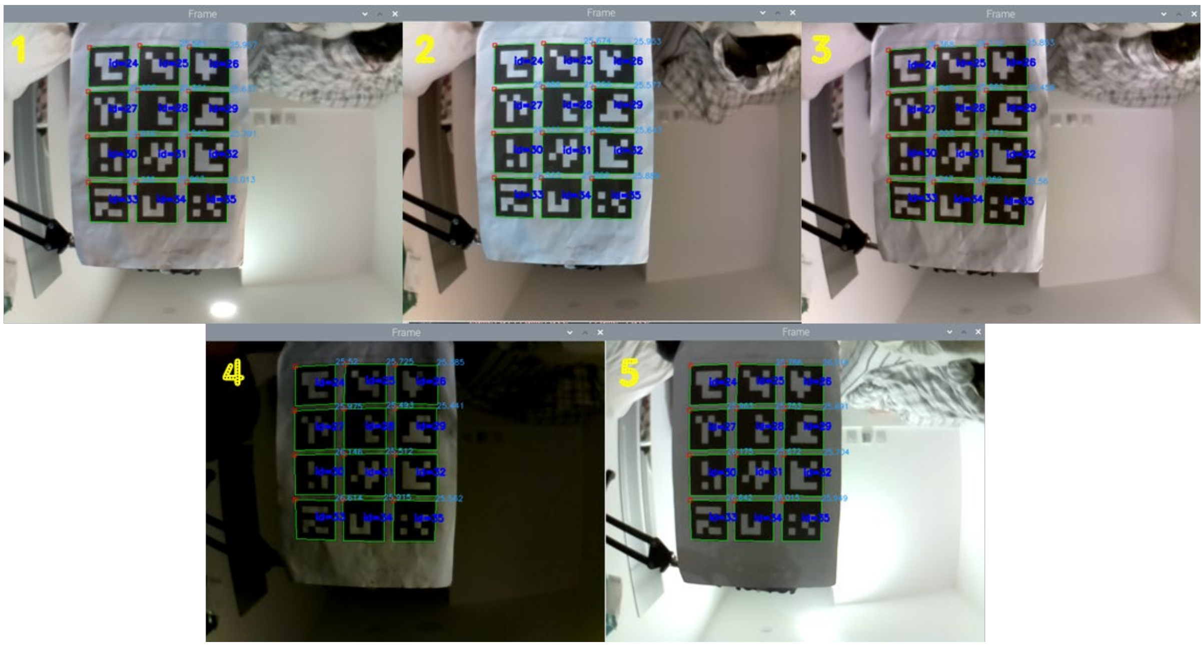


Figure_6.2: The result of different lighting condition.

From figure_6.2 the testing result is very impressive, the success detact all 12 markers even in the very dark condition and very strong backlighting condition, the algorithm can still works. However the condition is not include out door result the robustly in out door is unknown. Also the testing set have some disadvantage in optics way. The testing is not moving and stock in certain distance. we know the camera to capture image, there is some parameter tat is automatic run by OpenCV, which is ISO sensitivity, the shutter speed. In general, the lower the ambient lighting to keep EV (exposure value) to make image is visible. The system will usually slower the shutter speed and increase the ISO. To slower the shutter speed will cause more motion blur will marker is moving. this will make image not clear and higher ISO will cause the image noisy also make the image worse. If the problem detect rate is low in some dynamic condition, we can adjust the camera ISO, EV, shutter speed to optimize the result.

2. **Experiment: The angle of marker in the fix ambient light. Hold the marker at:**
   To finds out the marker visibility in extreme condition is also the way to measure the system robust. This is the expand of the first experiment. The same testing space Figure_6.1. and the light setting is 1600 stand light 2000 up room light.
   a) Face 90 degree
   b) Face 75 degree
   c) Face 45 degree
   d) Face 30 degree
   e) Face 15 degree

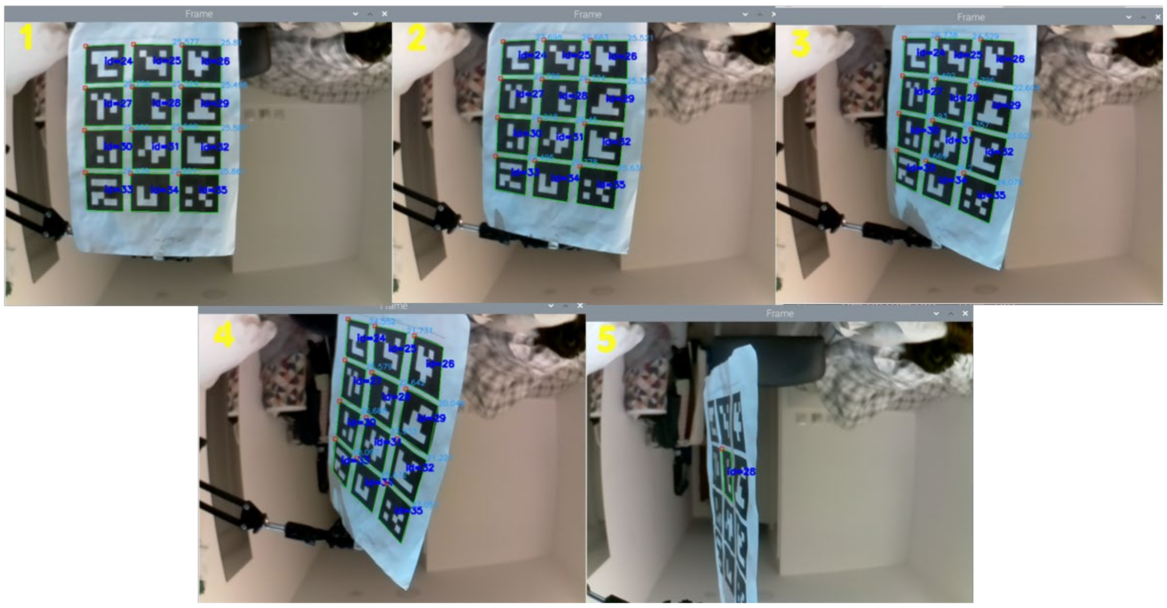


Figure_6.3: The result of different angle face to the camera.

From image_6.3 the test result is also impressive. The only failure case is 15 degree face to the camera, however it still can detect one marker in the image. although the testing result is good, but the disadvantage is same as the first experiment. The dynamic motion and the lighting in nature will make detecting task more complicate. The optimize is also can see *detect marker section*.

3. **Experiment: Feature Marker ID sorting validation**

Marker ID list at each detection epoch is in random order. Random order is due to the instability of environment the camera is located at: light, marker positions and background noise, etc... To ensure the target ID tracking in the random list at each epoch to support further path prediction computing, marker ID is sorted based on Euclidean distance from the camera.

To test the feature of sorting correctness, orient the test set (See Appendix A) clockwise and anti-clockwise respectively in different inclination.

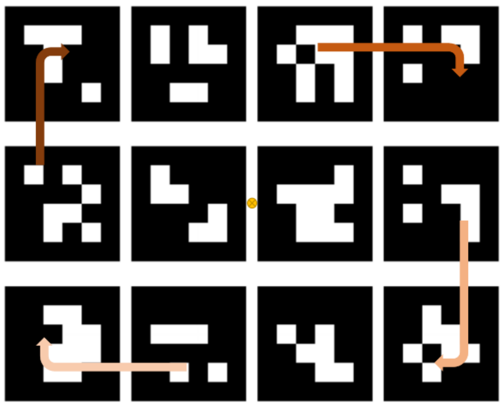
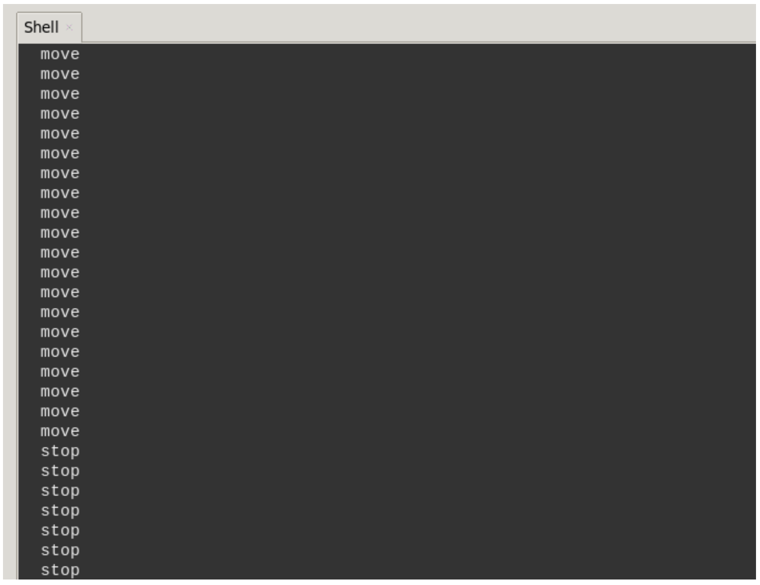


Figure_6.4: The spin example of test set.

The result can see the in the video demonstration. From the test result, it looks goods, the closest marker can be location draw in the edge of test set. The issue tat affect is also the camera parameter discus previous experiment problem, also the resolution can be the one of main issue tat can affect the successfulness of detecting markers.

4. Experiment: Path prediction validation when detection failure:

Path prediction feature can be validated when marker detection fails, this is the validation feature of the failure for the *task manager*. Failure of detection may due to influence of ambient light(background color disturbance or extreme ambient light). The robot is designed to reach the next target directly once the failure happens. The IO pin config turns from left-right control to straight move and stop at the predict target. To set up the test environment, keep the fix ambient light and move the marker toward camera. Put a piece of blank paper suddenly in front of the camera to trigger detection failure. Example of serial terminal output of the test is shown in Figure_4.12(b).



```
move
move
move
move
move
move
move
move
move
move
move
move
move
move
move
move
move
move
move
move
stop
stop
stop
stop
stop
stop
stop
```

Figure _6.5. Terminal output result the decision of task manager dose when detection failure

5. Experiment: Resolution and the time

The time cost for resolution is linear. The detection the ID of markers is the segment the marker image into the different grid. The less solution will cause the detection rate decrease. From the experiment is to finds the large performance and the high accuracy as the program default resolution. The accuracy is calculate by $Accuracy = \frac{num\ of\ detact\ marker}{the\ toatl\ num\ of\ detact\ marker}$

Resolution	Accuracy	Ave time
1920*1080	1	1.64
1280*960	1	0.81
640*480	1	0.28
256*144	0.6	0.08

Table_6: the resolution Accuracy, time cost test. From above the resolution 640*480 is the good resolution we can use in this project.

6. Experiment: Relation between number of markers detect and the epoch process time

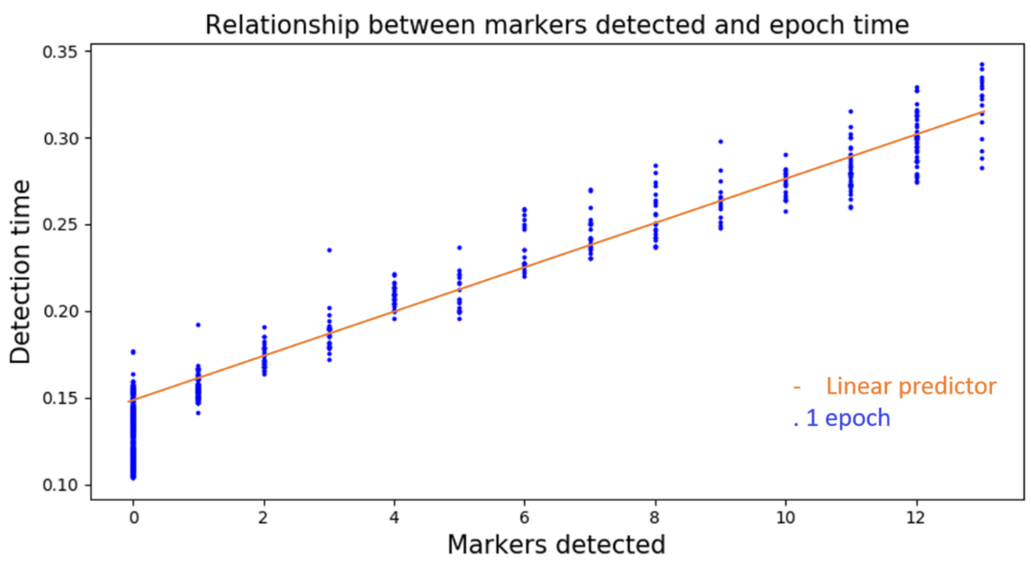


Figure_6.6: Markers detected versus time spent on detection.

Figure 6.6 illustrates the relationship between markers found and the respective detection time. The data in the figure was obtained in random light conditions for testing the marker detection strength of program. Software validation under different lights condition is mentioned in *experiment 1*. In the figure, every blue dot represents one epoch of detection process. The largest number of markers detected in the data is 13 during the data collection stage. The plot shows a linear relationship between markers detected and detection time, where the linear predictor is visualized in orange. Maker equals 0 means either the program fails to detect, or the program is processing other task. Even the program is not in detection epoch(when $marker = 0$), the task performance time is shorter than the time taken for at least 1 marker is detected in average(when $marker \geq 1$). Overall, the respond time for our detection algorithm is less than 0.35 seconds as shown in the figure. The average detection time with each number of markers detected from 1 to 13 is listed in the table below:

Markers Detected	Average Time(s)
1	0.142
2	0.152
3	0.171
4	0.199
5	0.201
6	0.212
7	0.231
8	0.246
9	0.25
10	0.268
11	0.284
12	0.299
13	0.331

Table_6.6: Average detection time under random lighting condition.

From table_6.6, the average respond time of each epoch is clearly listed. This indicates that the computational time for the algorithm is satisfactory. In the future, the time taken for detection using mor markers could be tested outdoor.

7.Result achievement

For conclusion The Project has develop base on the embedded computer vision with the Aruco marker in the Raspberry Pi w platform, develop software feature aiming with the ELEC2209 module target needs, by building up the robustly, easy use system. The program has designed the feature of Task manager, the methods for path panning, to face several different cases may happen in the usage. And setting up some experiment to test the feature and the system performance. The software design and the produced result satisfy the requirements of the project. Due to time limitation, the project successfully implemented the guidance systems without a web interface.

7.1 Analysis

During the development of the project, we found out some issue will significantly affect the testing result. The Resolution issue, the lighting environment issue, The photo binning issue, the focus of camera.

The effect of binning

The camera matrix has the fixe resolution, it have the equal maximal image resolution. The image in 1280*960 the pixel have 1.2MP in the image, The 640*480 have quarter of resolution. 0.3 MP the pixel has 4-> 1 pixel transformation the binning problem is appear on the edge of marker as the marker shows in fowling graph.

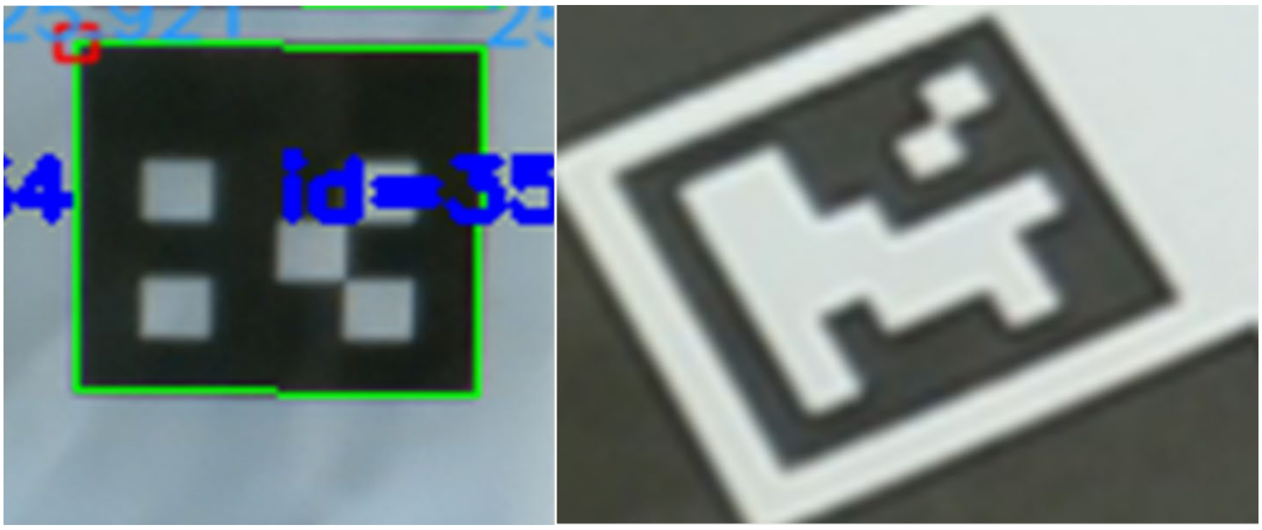


Figure_7: The photo taken from raspberry pi camera. Zoon in we can see the edge binning problem.

The focus problem

As the camera have the fix focus length, to adjust the camera by spin the camera lens see *camera calibration* . before doing any image processing pls make sure the camera is in focus of the range in markers. The effect impact is huge.

Lighting condition
From the test and validation section we can see the lighting environment is also impact the result of the output. The way to optimize the process of the detecting marker we can adjust the parameter in table1. It will cause the processing time change and the successful rate amount the different lighting.

7.2 Improvement

The program of the project currently is not perfect due to some of the issue these will be discussed in the following

Extrapolate feature:

This feature is to predict several markers path to get the better path planning. However Due to the coordination of the camera is local perspective to the camera, the localization for the marker is always moving in the local localization. And we only have one camera as the input sensor, it is challenging to complete robot localization globally. The program only detect the marker as the only reference object, if the markers is not in the frame there is nothing to reference. In the future more senser may be added as the other reference. Which can be use more advance path planning.

Marker vision limitation

In this project we only detecting the marker in the image, in the real world there are lots of unknow issue happens every time. By only detecting marker in the image is disadvantage of when detecting is failure. Also is limit for tracking more object to localize the robot and marker. In the future the use of object detecting to recognize the track directly instead of detecting markers will more reliable.

Failure detection decision making:

When the tracking target is losing the program will trigger the failure decision making. For current program, the task manager is update the last time robotic speed and the arrive time depends on the robotic speed. By only depend on calculation for estimate whether the robot is arriving or not is not accurate. The error, drifting of robot is happens every single time, to improve the prediction will need add more reference object or data, such as the IMU gyroscope to getting the gesture or any movement happen in the robotic or using the image to recognize more object to recognize the object in the environment, however it is compute expensively.

Reference

Ettlin, A. & Bleuler, H. (2006) Rough-Terrain Robot Motion Planning based on Obstacleness. In: 2006 9th International Conference on Control, Automation, Robotics and Vision. [Online]. 2006 IEEE. p. Available from: doi:10.1109/icarcv.2006.345116.

Sagitov, A., Shabalina, K., Sabirova, L., Li, H., et al. (2017) ARTag, AprilTag and CALTag Fiducial Marker Systems: Comparison in a Presence of Partial Marker Occlusion and Rotation. In: Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics. [Online]. 2017 SCITEPRESS - Science and Technology Publications. p. Available from: doi:10.5220/0006478901820191.

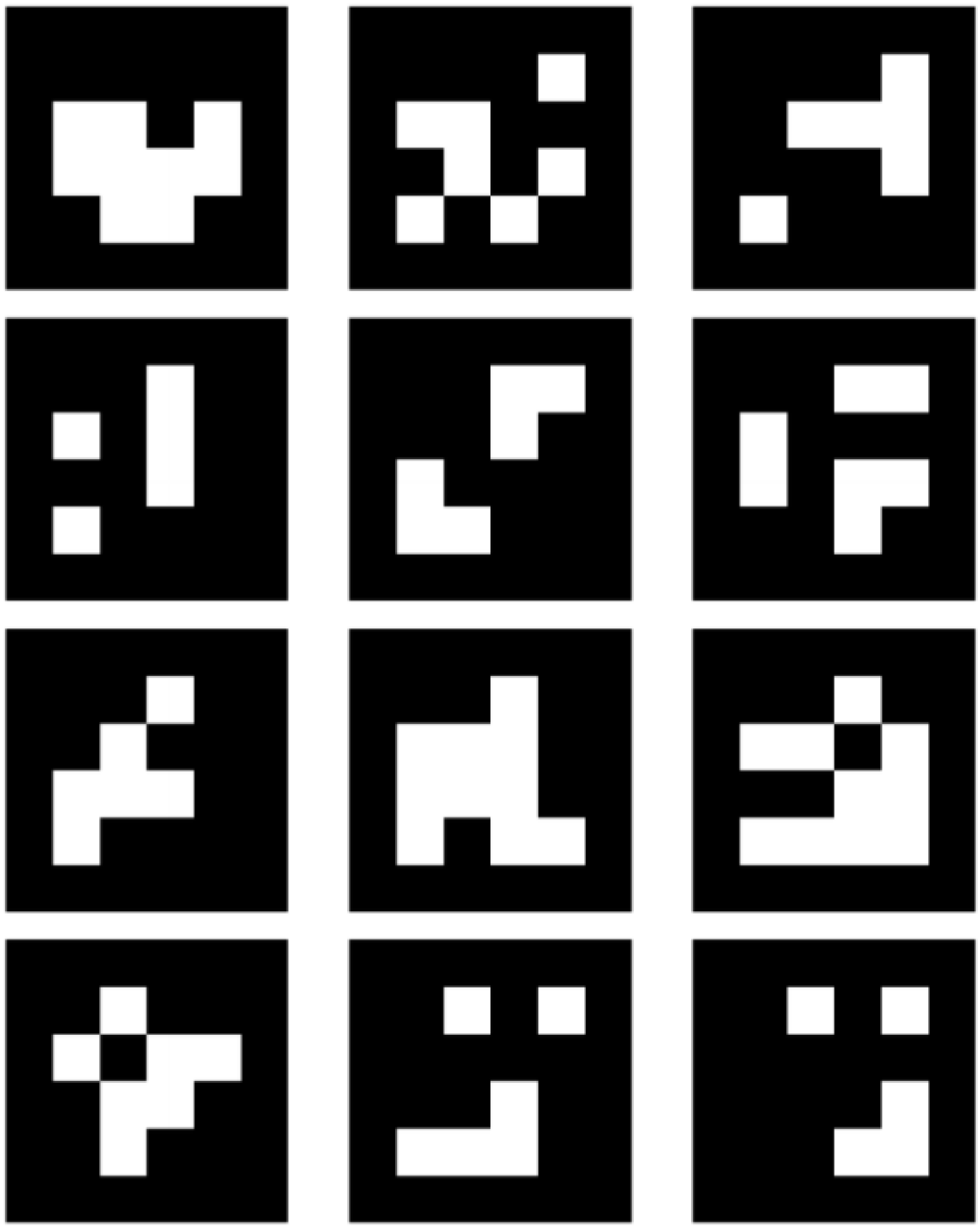
Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F.J. & Marín-Jiménez, M.J. (2014) Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition. [Online] 47 (6), 2280–2292. Available from: doi:10.1016/j.patcog.2014.01.005.

Kroeger, O., Huegle, J. & Niebuhr, C.A. (2019) An automatic calibration approach for a multi-camera-robot system. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). [Online]. September 2019 IEEE. p. Available from: doi:10.1109/etfa.2019.8869522.

Khan, U.A., Kar, S., Sinopoli, B. & Moura, J.M.F. (2008) Distributed sensor localization in Euclidean spaces: Dynamic environments. In: 2008 46th Annual Allerton Conference on Communication, Control, and Computing. [Online]. September 2008 IEEE. p. Available from: doi:10.1109/allerton.2008.4797580.

Appendix A

Test set: a 4x3 aruco marker



Appendix B

```
cv::aruco::PREDEFINED_DICTIONARY_NAME {
 cv::aruco::DICT_4X4_50 = 0,
 cv::aruco::DICT_4X4_100,
 cv::aruco::DICT_4X4_250,
 cv::aruco::DICT_4X4_1000,
 cv::aruco::DICT_5X5_50,
 cv::aruco::DICT_5X5_100,
 cv::aruco::DICT_5X5_250,
 cv::aruco::DICT_5X5_1000,
 cv::aruco::DICT_6X6_50,
 cv::aruco::DICT_6X6_100,
 cv::aruco::DICT_6X6_250,
 cv::aruco::DICT_6X6_1000,
 cv::aruco::DICT_7X7_50,
 cv::aruco::DICT_7X7_100,
 cv::aruco::DICT_7X7_250,
 cv::aruco::DICT_7X7_1000,
 cv::aruco::DICT_ARUCO_ORIGINAL,
 cv::aruco::DICT_APRILTAG_16h5,
 cv::aruco::DICT_APRILTAG_25h9,
 cv::aruco::DICT_APRILTAG_36h10,
 cv::aruco::DICT_APRILTAG_36h11
}
```

Above dictionaries/sets are from OpenCv: https://docs.opencv.org/master/d9/d6a/group__aruco.html