A Tracking Robot By Using Turtlebot

By

Yuanyue You

An Honour report submitted for the Degree of:

BEng in Electrcial and Electronic Engineering

Abstract:

With the rapid development of robot vision, robot vision can be used in numbers of parts and using robot vision to do tracking task is an important orientation. Applying this technology can also be much useful in, such like, close observation of animals, secret tracking of suspects in criminal investigation.

So this project is designed as a tracking robot by controlling the turtlebot robot receive images of the camera and command the robot move to keep tracking the target.

So here, the robot require at least two parts the moving part which can move around with wheels and base, and the camera part which can capture the video surrounding the robot with the camera. According to the input image information, and combining with yolo algorithm (You Look Only Once) to detect the objects which are shown on the screen by camera video input. At the same time, the screen should also show the detected object class and the track id etc. These information can directly convenient the user observe the video shown on the computer screen delivered by the camera and then select which detected objects as the tracking task target.

Then, after detecting the objects and the camera can recognize such objects, next step is to 'tell' the robot about the distance from the detected objects and turtlebot by using the depth information supported by the camera. Because the turtlebot needs to track the selected object automatically. Then the turtlebot knows the distance and starts to move forward or drawback to keep a distance with the detected target which is the tracking task starting.

Acknowledgements

Many people give me valuable assist in my thesis writing and project processing include my supervisor and my teammates.

First of all, I would like to show my sincere gratitude to my supervisor Seng Wang, he really helped me a lot not only on the 4th year project but also the career planning. He asked us that as a student, our main job is study. Learning new knowledge, and getting improved during college life is essential and only in this way, it is worth that we spent a lot of money to study in the university as a student. Besides, this period is a very good timing to learn something useful, because we can focus on studying and have enough energy to absorb new knowledge. We should improve ourselves as much as possible during this period, because if we start to work later years, we may not have enough efforts and such excellent learning environment. And my supervisor focus on the project, he concern about our project process, but he care much more about our questions we encountering during doing our own project. So each time of meeting, we spent most of time on how to solve various of problems. According to his numerous robot knowledge and experience, his advice always was useful and help us understand how to finish the total project.

Second, I want to give my thanks to my teammates, Kaicheng Zhang, Ziyu Du, Zhanheng Li and Zhuocheng Han. Doing a new project is always tough at the beginning time, because we need to grasp large amounts of knowledge that we may never learned before. Thanks to my teammates, we built a group and shared our learning materials. We helped each other and improved ourselves at the same time. Besides, we usually met in library to communicate with each other about our problems and gave the advice. Of course, there always were some problems that all of us can not solve, then we would ask our supervisor during the meeting. But anyway, we still solve many problems and this group and my teammates helped me a lot.

Statement of Authorship

I, Yuanyue You

State that this work submitted for assessment is my own and expressed in my own words. Any uses made within it of works of other authors in any form (eg. Ideas, figures, text, tables) are properly acknowledged at their point of use. A list of the references employed is included.

Date: 2020/4/15

Index

Figure Index:

1. Introduction

1.1 Objective and aim

This project is to enable a robot to do the object detection and object tracking task. And hence, the robot must acquire at least two parts, the moving base which can enable the robot move around and the camera which is the input of image flow.
First, for the object detection function, it refers to the robot recognition technology. It means that the robot or the algorithm need to understand what the object belongs to in front of the robot camera. According to the input image flow by camera, the algorithm calculate the received data, and then return the position in the screen, and the distance between camera and the detected objects.
Second, for the object tracking task, the robot should follow the detected object and at the same time the object detection is also needed. Robot should move forward or drawback, turn right or left to keep a selected distance with target.

1.2 Project description

According to the project's objective, it is convenient to apply some models and algorithm to finish parts functions of this project.
First is the project developing environment, this project is ROS supported, all the functions are satisfied in ROS. And the main part used in ROS is the topic function which is used for communication with different plugins.
Second, the project applies the turtlebot2 model to satisfy the robot movable and image flow available for algorithm. The turtlebot2 model is consist of the moving base and kinetic camera. These two main parts support most of turtlebot2 functions such like location, tracking, obstacle avoidance and so on. The turtlebot2 is a ROS dependent robot platform. So for the kinetic camera, it publish the image related information topic such like /camera/rgb/image_raw, /camera/depth/image_raw topic, and the project mainly use these two topic as the object detection and object tracking tasks' algorithm input data. Besides, the turtlebot moving base will move around according to receiving

the commands by subscribing the /teleop topic.

Third, for the object detection part, YOLO algorithm is used in this project. And the newest version is yolov3. The yolov3 is used to detect the object according to the continuous image frames then it will return the class of detected object, the confidence, the coordination in the frame. For example, if there is a person standing in front of kinetic camera, the kinetic camera will send continuous pictures to the yolov3 algorithm. Generally speaking, the robot do not know which part is the person, it can only receive numbers of pixels information matrix which include each pixels color information, here is rgb(red, green, blue) information. Then the robot uses yolov3 algorithm to get all the pixels information, and do the detection, then the robot knows there is a person standing in front of it. The confidence is what confidence this algorithm has to regard the object in the picture is a person. And yolov3 will also return a coordination with (rows, columns) the rows and columns are the number of pixels in a picture. The rows-th pixels and columns-th will cross one-pixel point. And the rows and columns are the coordination of this pixel.

Then the detected person coordination which is the returned pixel coordination is known, the tracking algorithm start to subscribe the /camera/depth/image_raw depth info to get the distance information of the person according to the returned pixel's coordination.

However, only the objection detection by yolov3 is not enough for the tracking task, because the yolov3 can only know the detected object class for each image frame, but it can not know what the relationship between the same detected objects in two continuous frames is. So here also needs re-id technology which can tell the robot the persons shown on the screen with continuous are the same person. And this technology can also solve the person disappearing suddenly problem. Because, there will be always some special occasion occurred, such like the detected person1 disappear from the screen suddenly. Some obstacle impedes the vision of camera or the person turn to the back of building due to the corner there, all these occasions can cause the person1 suddenly disappear. So here also needs the re-id technology.

By now, the person is detected according to the yolov3, and the distance from people and camera is also know by the depth info topic. Then the tracking task can start. The user is acquired to input a desired distance which user hope the robot keep such distance with the person. And user is also needed to select a detected object from all the detected objects shown on the screen as the target. The detected objects shown on the screen is something like the figure 1.1 shown below.

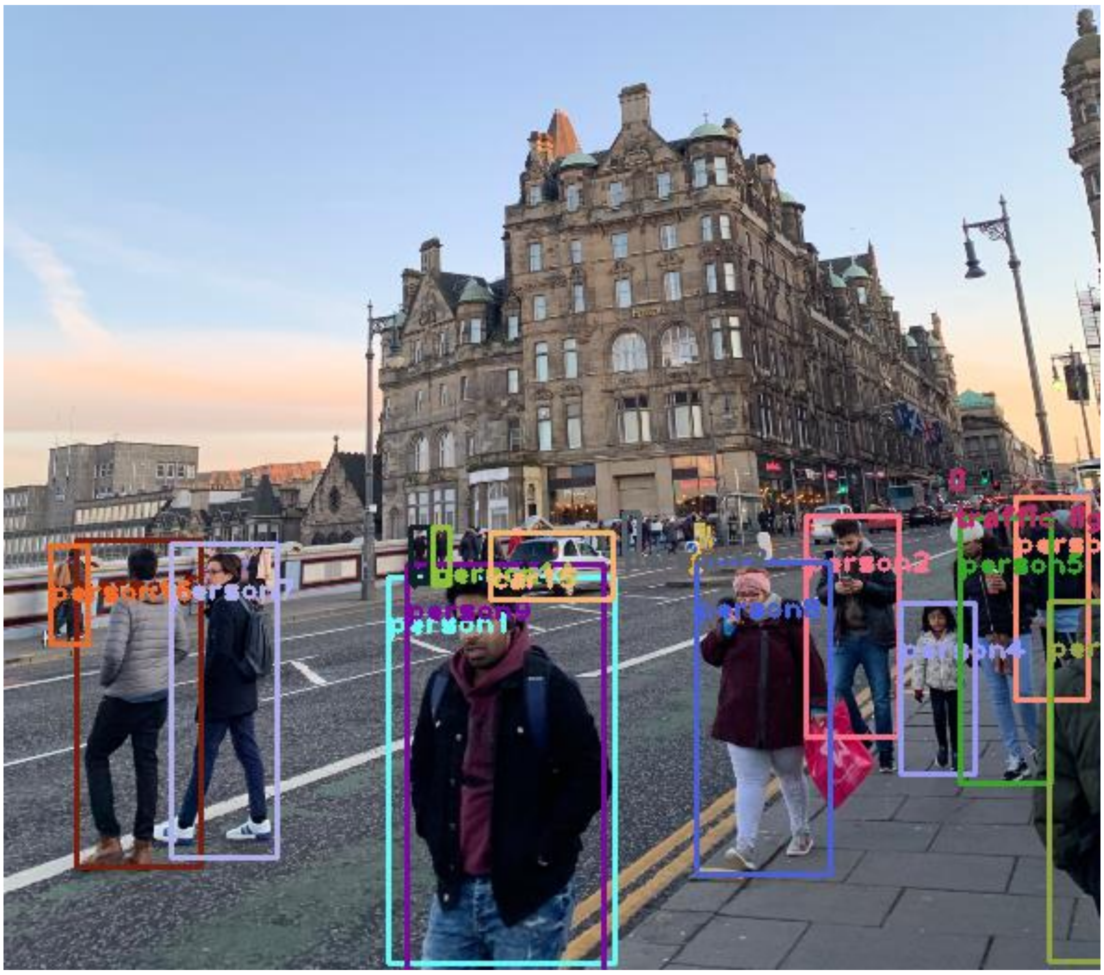


Figure 1. 1 Detected objections with class name, track index

And user is required to input the class name of the track target, for this example is "person", and the track index. Because, there may be more than one person shown on the screen, and the track index can specify different person and for this example, the track index is set 1.

Finally, according to the information of class name, track index, desired distance, real distance between person and robot, and detected person coordination, then the turtlebot start to track person 1. The robot will move forward or drawback to try to keep the desire distance with the person 1 and turn right or turn left to keep the person 1 always shown on the center of screen. This procedure makes sure the detection actuary all the tracking time.

In all, the camera sends the color information to tracking algorithm, yolov3 as the part of the whole tracking algorithm is used to detected object and return the class name and the pixel coordination. Then re-id technology combines the two detected objects in two frames together and tell the tracking algorithm they are the same. At the same time, the tracking algorithm subscribe depth information, and find the distance between camera and detected objects. Finally, it publish the /teleop_cmd_vel topic to control the turtlebot to do the tracking task. All the procedures can be shown in the flow chart, figure 1.2 shown below.

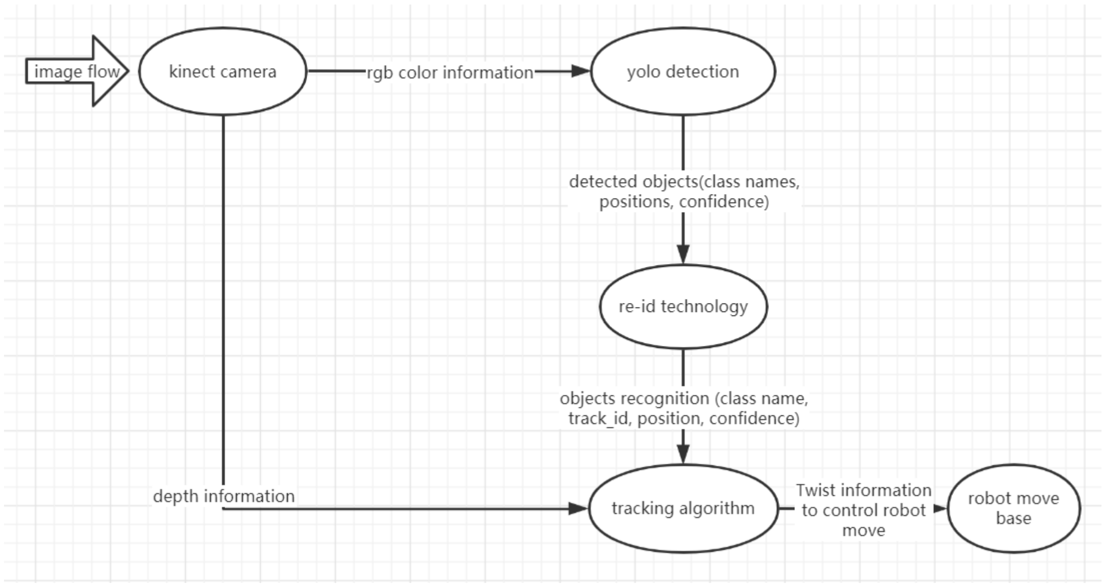


image flow
kinect camera
rgb color information
yolo detection
detected objects(class names, positions, confidence)
re-id technology
objects recognition (class name, track_id, position, confidence)
depth information
tracking algorithm
Twist information to control robot move
robot move base

Figure 1. 2 flow chart of all the tracking system procedures

1.3 Application

This project can be used in various areas in for tracking task or the object detection task.
It is convenient to do such like closed distance observation of wild animals and tracking them to search their daily life style and even find their habitat where generally people can not easily arrive or these animals will be threaten and warned by using little animals modeled and moveable robot with this project technology. Besides, this project can also be used to calculate the population flow at some flourish square for a specified time. Moreover, it is convenient to do the secret tracking of suspects in criminal investigation by using small scaled tracking robot.

1.4 Introduction of each chapter

Here is the brief introduction of what the next chapter mention about, and this introduction can be convenient to better understand the logic of this report, and have a clear clue that how this whole project is finished.
Next chapter is chapter 2, background, this chapter mainly describe the background technologies which are used in this project, some description of ROS, YOLO object detection algorithm and the turtlebot models.
Then turn to chapter 3, process and result. This chapter will shown how the totally project is finished, and how this project applies the technologies mentioned above. This part will show all the procedures step by step and some problems encountered during the developing period.
After finishing all the functions of the tracking robot, then the conclusion and estimation is the main content of chapter 4. Here will do a summary about all the project, and estimation about which parts are needed to be improved and the project's weakness.
And chapter 5 is the reference of all the materials and literature. How this project come from and how to solve problem by reading books or thesis. All the citing parts are shown in chapter 5.

Chapter 6 is appendix which includes the meeting notes for the two semesters period between students and supervisor.

2. Background

2.1 ROS (Robot Operating System)

2.1.1 Introduction of ROS

ROS is an open source meta operating system for robots. It can provide many functions similar to the traditional operating system, such as hardware abstraction, underlying device control, common function implementation, communication between different threads and package management. In addition, it also provides tools and libraries for acquiring, compiling, editing code, and running programs among multiple computers to complete distributed computing.

Because of ROS, the robot development is much convenient than before. ROS is a framework for robot programming, which provides a communication framework for different plugins and sensors. Although ROS is called an operating system, it is not a common operating system like windows and MAC. It just connects the operating system and the ROS application you developed. So it is also a middleware or a platform. The application based on ROS establishes a bridge of communication, so it is also a runtime environment running on Linux. In this environment, robot perception, decision-making Control algorithm can be better organized and programmed.

ROS supports multiple programming languages. C ++, pyhton and have been compiled in ROS, which is the most widely used ROS development language at present. The test libraries of lisp, C# and java have also been implemented. In order to support multi language programming, ROS uses a language neutral interface definition language to realize the message transmission between modules. The common understanding is that the communication format of ROS has nothing to do with which programming language to write. It uses a set of communication interfaces defined by itself.

Currently, ROS is only supported on Linux system for installing and config developing environment. And the first choice for installing ROS is the ubuntu platform. By now, ROS has updated several version for different ubuntu version developers using. And each version of ROS has their own recommended ubuntu version developing environment. And for now, ROS Kinetic and ubuntu 16.04 are most popular combination and also there are most numbers of learning materials for this combo. So in this project, the Kinetic and ubuntu 16.04 are selected as the environment of developing.

2.1.2 Communication in ROS

ROS has more convenient functions and more communication methods such like service and action. But in this project, only the topic is used, so here only introduce the used technology and some definition in ROS. If more information is needed, please check the ROS wiki website which is http://wiki.ros.org.

(1) Node and master

Node is the smallest process unit in ROS. For a project, there may be many executable files to finish the whole task for a series of process flow, and after running executable file, there will be a process running, and this process is called as node in ROS environment. So the node is an executable file which generally generated by the compiling C++ code and a python script. And the node is responsible for one function of the robot. Due to the complexity of a robot function, usually there needs more than one node to cooperate with each other. And this method is also convenient for debugging and has very clear division of task. Just imagine writing all the functions in one processing, there will be quite troublesome to deal with the communication between models and debug.

Now that there are so many nodes existing in one project, and they need to cooperate with each other. Each of them havs their own tasks such like controlling the movement of robot, object detection, calculating and so on, then how they are managed. So here the master comes out. Master is actually a node manager. One node can work only after registering at master, generally by using the code below to initial node at master in python.

rospy.init_node('node_name')

Then the master will lead the new node to the ROS and the communication between two nodes are also connected by master. So from the relationship between master and node, it is essential to activate master first, and then the node start to register and work. The relationship can be more clear by the figure 2.1 shown below

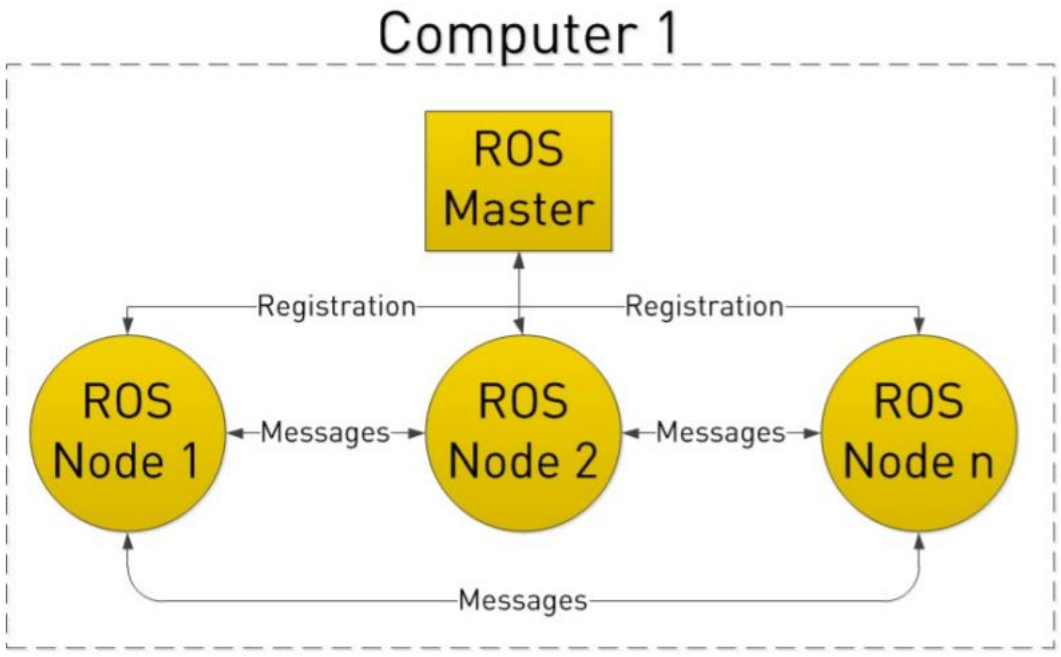


Figure 2. 1 The relationship between master and nodes in ROS

(2) Topic

Topic communication is very common in ROS. Topic publish or subscribe the real time and period messages, and topic is transmitted between two nodes. There are some steps for initializing topic. As mentioned before, the topic is transmitted between two nodes, so one node is set as publisher, the other will be set as subscriber. Besides, publisher and subscriber all need to do the registration at the master, then the publisher will publish topic, and the subscriber will listen the topic by the management of master. Then the topic communication bridge is built between publisher and subscriber. The communication process can be shown in Figure 2.2

shown below.

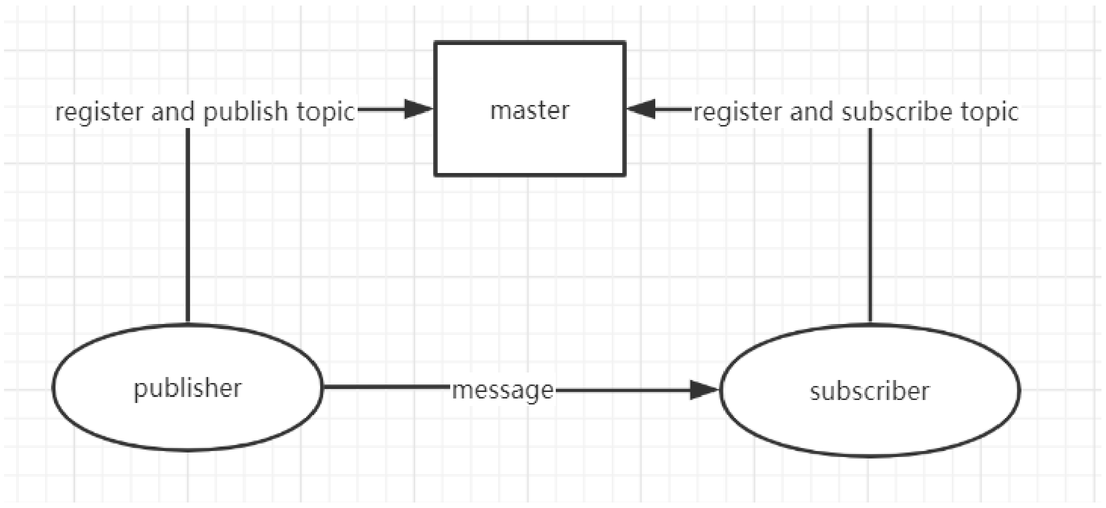


Figure 2. 2 Topic communication process

At the subscriber terminal, there is callback function to deal with the data received via the topic, and this callback function is needed to be written in the code before the subscriber start to listen topic. Once the subscriber listens the topic, the callback function will deal with the received data. On the other hand, one topic can be subscribed by multiple subscriber, and also the same topic can be published by more than one publisher.

(3) Message

Topic has very strict format requirements. For example, in the process of camera, the topic of RGB image must follow the RGB image format defined in ROS. This image format is message. Message is the data type and format standard of topic content.

The basic msg includes bool, int8, int32, int64, float32, float64, string, time, duration, header, array [] type of data. And generally, these msg type is located in std_msgs, sensor_msgs, geometry_msg. The three message types are also used in this project.

Here are some message used in this project.

Twist.msg (this msg is used to control the movement of turtlebot)

#define the linear velocity and angular speed of object

#location: geometry_msgs/Twist.msg

Vector3 linear

Vector3 angular

This Twist msg is used to control the movement and rotation of the object. As shown in the msg, there are totally two elements, the linear, and angular. For Vector3, this class include x,y and z three components in float 32 type. Each component control the speed in corresponding direction. For linear movement, below Figure 2.3 shows the direction of linear movement. For example, if linear.x = 1 , linear.y = 0, linear.z = 0, which means that only x component has values, then the object will move along the x axis straight forward with the speed is 1.

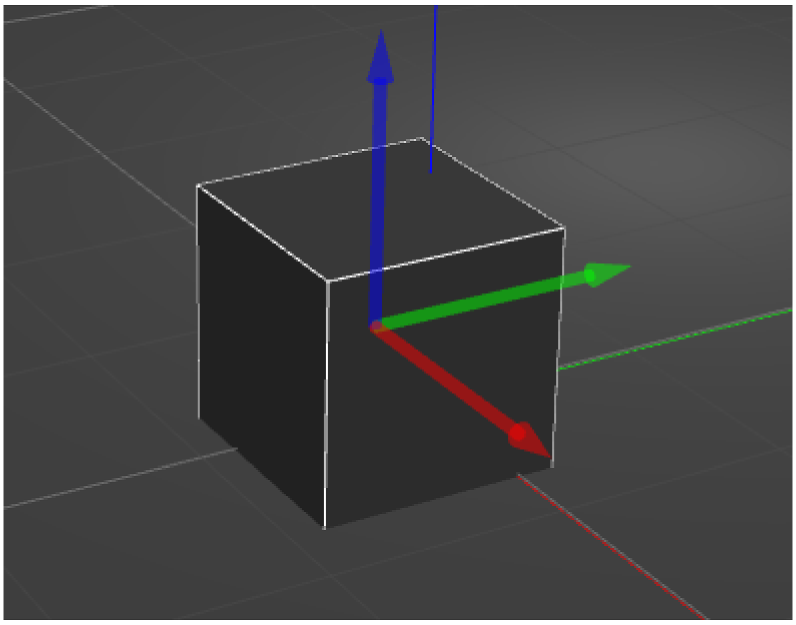


Figure 2. 3 The linear coordinate of the box, the x, y and z are red, green and blue respectively

For the angular turn, there is also an angular coordination of x, y and z. however, this coordination is not same as that of linear system. For example, if the angular.x = 1, angular.y = 0, angular.z = 0, the object will rotate with the center of x axis. Figure 2.4 shows the angular coordination.

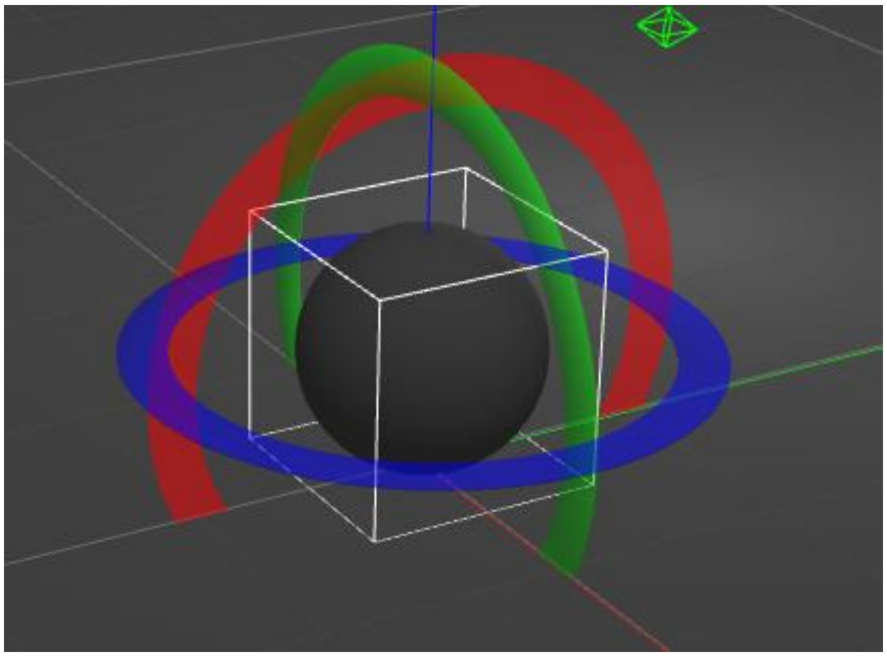


Figure 2. 4 Angular coordination, object will rotate with the center of axis, the x, y and z axis are red, green and blue respectively

So for this project, the turtlebot2 is used as the robot model, turtlebot can move with linear movement in the horizontal direction which are x and y direction. And it can turn around with horizontal which means that only angular turn with z axis has values.

```
Sensor_msg/Image
Std_msg/Header header
    Uint32   seq
    Time     stamp
    String   frame_id
Uint32  height
Uint32  width
String  encoding
Uint8   is_bigendian
```

Uint32   step

Uint8[] data

This Image.msg is the msg type used in subscribing the color information and depth information from /camera/rgb/image_raw and /camera/depth/image_raw respectively. Tracking algorithm subscribe these two topic to do the object detection and measure the distance.

2.2 YOLO (You Look Only Once)

2.2.1   YOLO introduction

Yolo (You Look Only Once) is an object detection algorithm and up to now this algorithm has update to yolov3, by using this algorithm, robot can recognize totally 80 kinds of objects such like person, bicycle, car, etc. And all these types of detectable objects can be found in the coco.name file in the darknet official website https://pjreddie.com/darknet/yolo/. On the other hand, if the included classes in the coco.name can not be supported for some specific objects, users can also train their own models and add these trained classes into the detectable classes

2.2.2   The logic of YOLO

The core idea of Yolo is to use the whole graph as the input of the network, and directly return the location of the bounding box and its category in the output layer. Firstly, the yolo algorithm only support the specified sized image which are 244*244, 416*416 and 608*608. So before using yolo algorithm, it is essential to resize the input image to the scaled size. User need to select a scaled size as the input image size, the 244*244 scaled size is fastest, but has lowest accuracy. 608*608 is the most accurate but slowest. Generally, the 416*416 scaled image is chosen as the input image, it has not so slow calculating speed and pretty good accuracy.

Specifically, Yolo's CNN network divides the input image into an S × S grid, and each cell is responsible for detecting the targets whose center points locate in the grid. The divided image shown in Figure 2.5

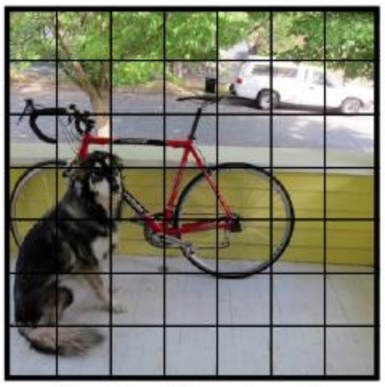


Figure 2. 5 The S*S grid on input of the image

Each cell will predict the number of B bounding boxes and the confidence score of the bounding boxes. The so-called confidence degree actually includes two aspects, one is the possibility of the target contained in the bounding box, and the other is the accuracy of the bounding box. The former is recorded as Pr(object). When the bounding box is the background (that is, it does not contain the target), Pr (object) = 0. When the bounding box contains a target, PR (object) = 1. The possibility of the bounding box can be used to predict the IOU ratio of the predicted frame and the ground truth which can be represented IOU_{pred}^{truth}. So the confidence degree can be defined as

$$\text{Confidence degree} = \Pr(\text{object}) * IOU_{pred}^{truth}$$

IOU is a function of object detection, Intersection over Union. And the IOU calculate the ratio of predicted frame and the real frame.

$$\text{IOU} = \frac{Intersection}{Union}$$

And the Figure 2.6 shows clear of definition of IOU

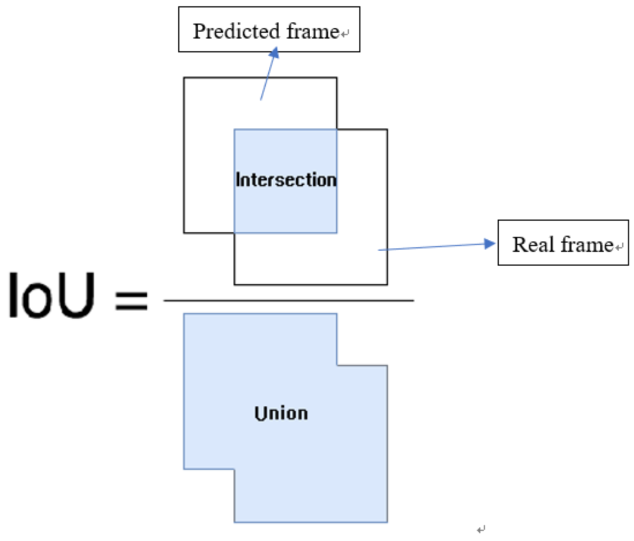


Figure 2. 6 Intersection Over Union

The size and position of the bounding box can be represented by four values: (x, y, w, h) where (x, y) is the central coordinate of the bounding box, and w and h are the width and height of the bounding box. It should also be noted that the predicted value (x, y) of the central coordinate is the offset value relative to the coordinate point in the upper left corner of each cell, and the unit is relative to the cell size. The definition of cell coordinates is shown in Figure 6. The prediction values of w and h of the bounding box are the ratio of the width and height of the whole picture, so theoretically the size of the four elements should be in the range of [0,1]. In this way, the predicted value of each bounding box actually contains five elements: (x, y, w, h, c). The first four elements represent the size and position of the bounding box, and the last one is the confidence degree.

There is also a problem of classification. For each cell, it is necessary to give the probability value of predicted C categories, which represents the probability that the goal of the bounding box predicted by the cell belongs to each category. But these probability values are actually conditional probability under the confidence of each bounding box, namely Pr(classi|object). So the class-specific confidence scores of each bounding boxes can be calculated by

$$\Pr(classi|object)*\Pr(object)*IOU_{pred}^{truth} = \Pr(classi)*IOU_{pred}^{truth}$$

So in all, each grid needs to calculate (B*5+C) values, and the total values is S*S*(B*5+C)

By now, there are numbers of bounding box with their coordination, confidence, width and height existed. Then the next step is filter the noise bounding boxes which are below a confidence threshold.

Here the non-maximum suppression, NMS algorithm is used. This algorithm is not only for Yolo algorithm, but also for all detection algorithms. NMS algorithm mainly solves the problem that a target is detected many times. First, find the box with the highest confidence from all the detection boxes, and then calculate the IOU of the remaining boxes one by one. If the value is greater than a certain threshold (the coincidence is too high), then remove the box; then repeat the above process for the remaining detection boxes until all the detection boxes are processed. Here is an example of the detections shown in Figure 2.7 below. And the Figure 2.8 shows the detection result after using NMS algorithm.

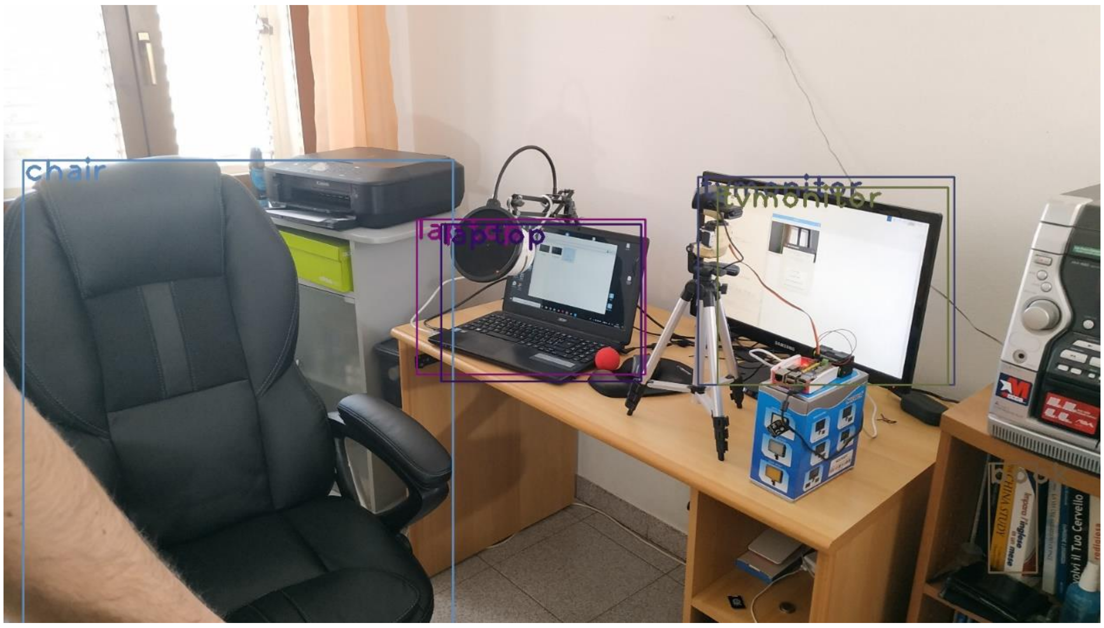


Figure 2. 7 The detection with same object is detected many times

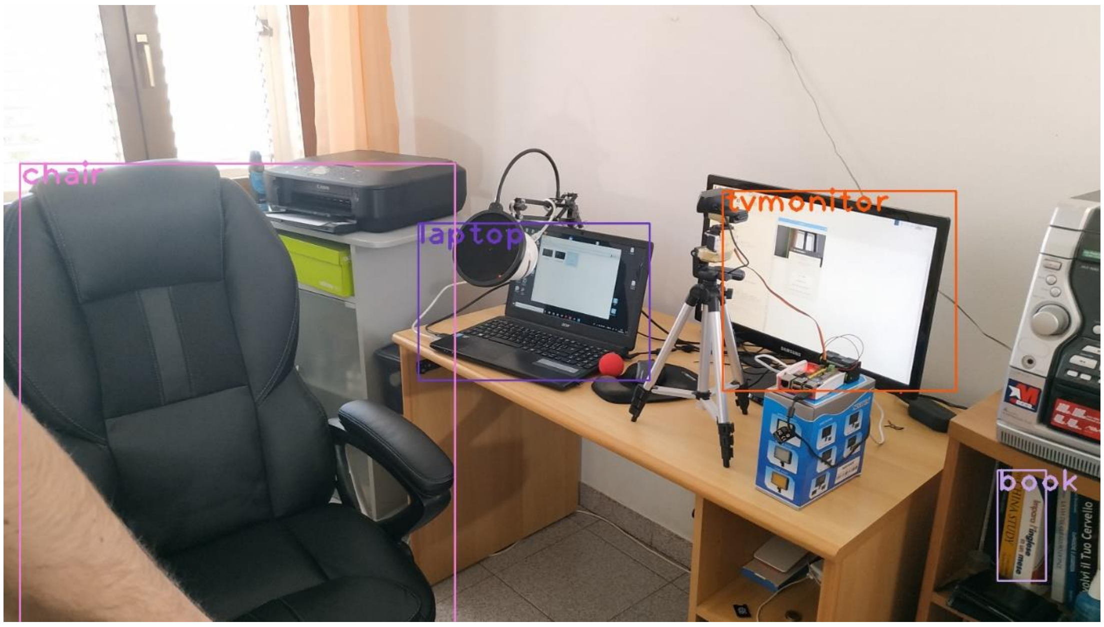


Figure 2. 8 The detection result after using NMS algorithm

After using NMS, the whole bounding boxes will be those which have most confidence degree. The whole process of YOLO can be shown clearly in Figure 2.9.

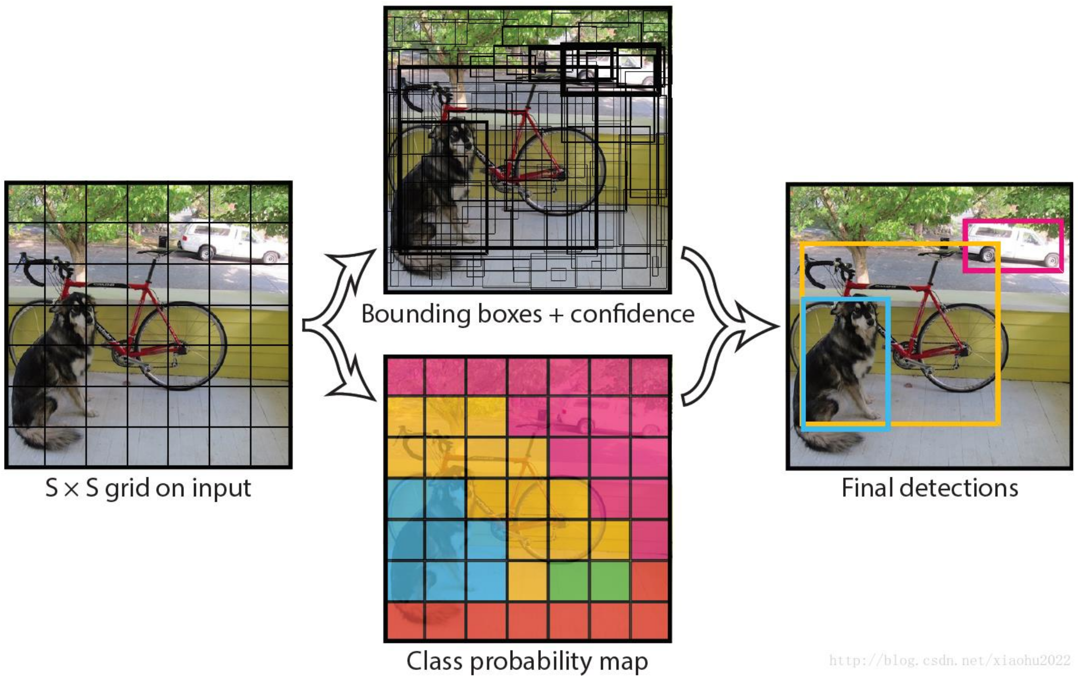


Figure 2. 9 The YOLO algorithm working process

2.2.3 The application of YOLO

YOLO now support not only for cpu and also for gpu, and the gpu has higher calculating speed which have higher FPS, the frames number per second. However, in this project, there is no GPU available, so the cpu is used for yolo detection. Besides, the project acquires opencv to deal with image input, so it is necessary to choose a suitable version opencv. From opencv 3.4.2, opencv start to support YOLO algorithm, and it has more API functions which can offer help to user for more convenient using of YOLO.

2.3 Turtlebot

Turtlebot is an open-source hardware platform and mobile base station. When ROS is used, turtlebot can deal with vision, location, communication and mobility problems. It can move anything on top of it to the designated place. Now turtlebot has several versions which include turtlebot, turtlebot2, turtlebot3 etc.

This project applies the turtlebot2 as the robot model so only the turtlebot2 is introduced in this report, if more information is needed, please check the turtlebot official website: https://www.turtlebot.com/ and the learning turtlebot website: http://learn.turtlebot.com/

There are several packages in turtlebot, and each packages has their own function, they cooperate with each other, and then the turtlebot can achieve the location, following, moving by keyboard functions and so on.

And there will be brief introduction of the function of part of packages, and the function which this project needs.

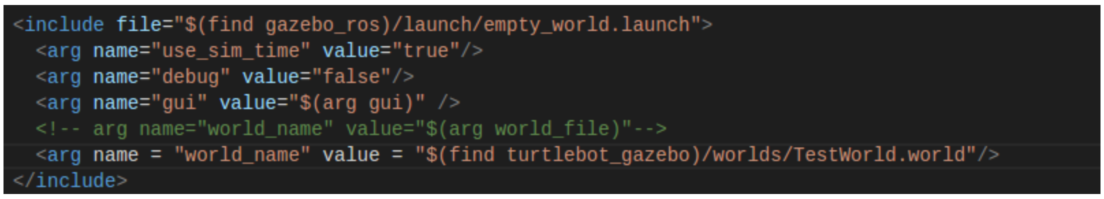
First, after download the whole turtlebot packages, there is a “turtlebot” folder, this folder builds the whole models of the turtlebot with the plugins, such like the moving base, kinect camera are all defined and created here. Besides in the turtlebot_teleop package, users can use this package to control turtlebot move around by keyboard. By using the command below

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

And then the introduction of how to control robot appears on the terminal. Using "u I o j k l m , ." totally 9 keys to control robot move and turn, besides q/z, w/x, e/c are also used by increasing/decreasing the linear speeds and angular speed.
This function is actually using different keys to change the values of Twist msg, then publish this changed Twist msg to the move base through topic /cmd_vel_mux/input/teleop

For the turtlebot_apps folder, this folder contains some superior functions such like the calibration, following, navigation functions, users can use these packages to finish these function in turtlebot robot.

Then turn to the turtlebot_simulator folder, one of the most important package is turtlebot_gazebo package in this tracking robot project. Users can lead the turtlebot model into the existed gazebo world, by using roslaunch turtlebot_gazebo turtlebot_world.launch. And users can change this launch file default world name to input their own world file. The part of code in launch file is the figure 2.10 shown below



```
<include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="use_sim_time" value="true"/>
  <arg name="debug" value="false"/>
  <arg name="gui" value="$(arg gui)" />
  <!-- arg name="world_name" value="$(arg world_file)"-->
  <arg name = "world_name" value = "$(find turtlebot_gazebo)/worlds/TestWorld.world"/>
</include>
```

Figure 2. 10 The turtlebot_world.launch file for input gazebo world

As shown in figure 2.10, user can use their own world as the testing world. Here the TestWorld.world is this project simulation world.
Then if all the turtlebot packages are installed and the workspace finishes catkin_make, then when running this turtlebot_world.launch file, there is master activate, and nodes, topics, services and actions are all enabled. Figure 2.11 is the result of running this launch file, the gazebo simulation world shown in Figure 2.11

below. And Figure 2.12 shows all the topics published in turtlebot. According to these topics shown on terminal, it shows that this turtlebot model works successfully.

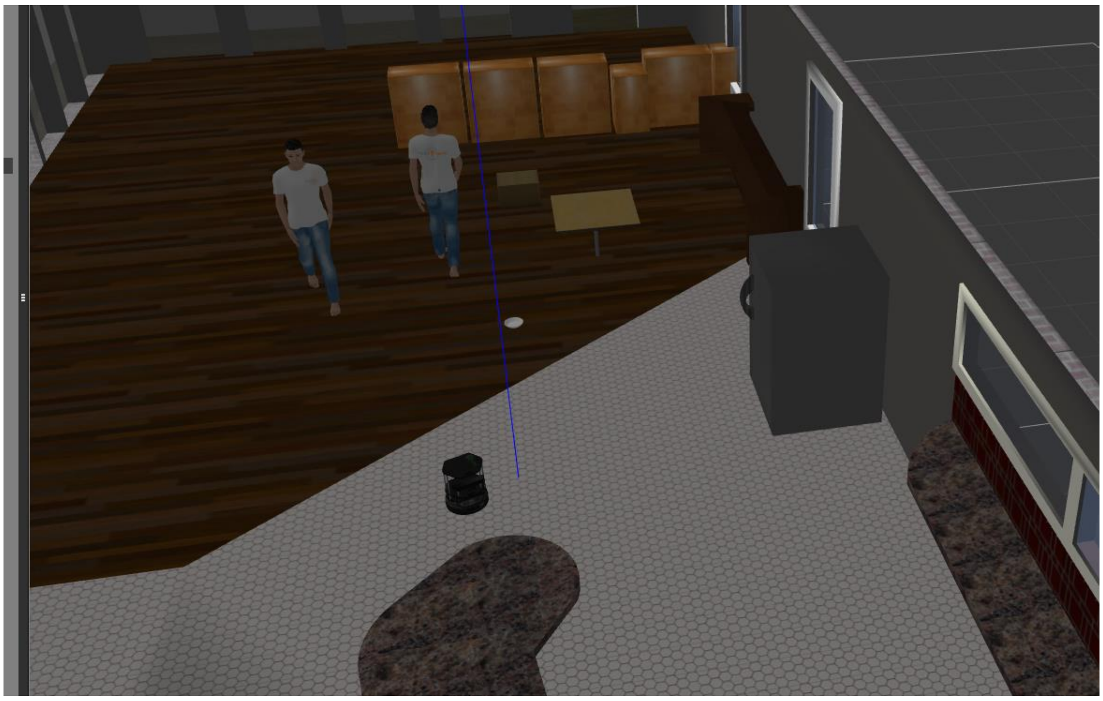
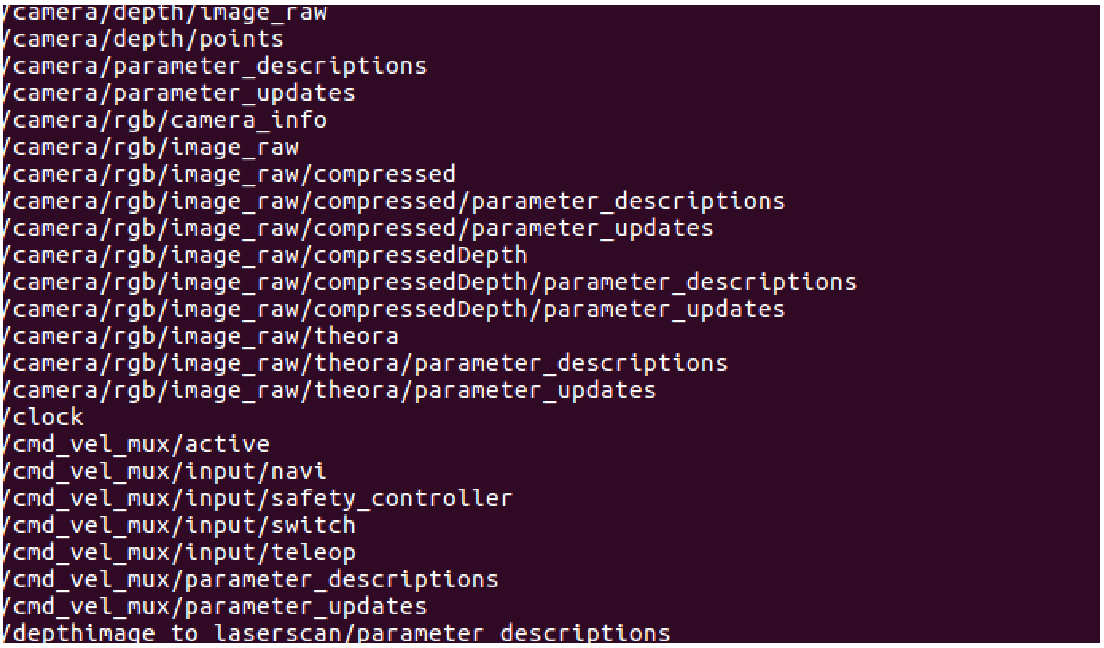


Figure 2. 11 The TestWorld.world simulation gazebo world with turtlebot



```
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/clock
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/switch
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/depthimage_to_laserscan/parameter_descriptions
```

Figure 2. 12 The topics published in turtlebot

2.4 Kinect camera

2.4.1 Kinect camera introduction

Kinect V1 depth camera has an RGB color camera, an infrared CMOS camera and an infrared transmitter. The camera's infrared CMOS camera and infrared transmitter are distributed in a left-right horizontal way. The camera uses the improved light coding technology based on structured light to obtain the depth information of the object.

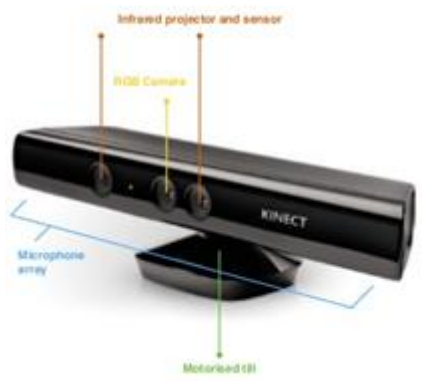


Figure 2. 13 Kinect camera

2.4.2 Kinect camera calibration

If the Kinect camera is needed in project, the first step is doing calibration and get the related camera info. Hence, the driver of Kinect camera is necessary, there are two popular drivers for Kinect camera, openni and freenect. However, after trying to use openni driver, the Kinect camera can not connect with computer for unknow reasons, so the freenect is the selected driver for camera. Use the command shown below to install freenect into ROS

```
sudo apt-get ros-kinetic-freenect-*
rospack profile
```

then use below command to activate freenect driver.

```
roslaunch freenect_launch freenect.launch
```

By now, all the preparation work is done, it is time for calibration. Install the calibration packages by using the command below

```
rosdep install camera_calibration
```

and run its python file to start calibration.

```
rosrun camera_calibration cameracalibrator.py --size 6x6 --square 0.03
```

image:=/camera/rgb/image_raw camera:=/camera

the calibration needs a black-white grid picture, and 6x6 is how much the grids this calibration needs to recognize, and 0.03 is the width of each cell in meter. The image receives the topic of camera rgb information. This calibration task will save different angle of grid picture (one example picture is shown in Figure 2.14) and do the calculation. And finally, the calibration result which is the camera info is got and shown in Figure 2.15

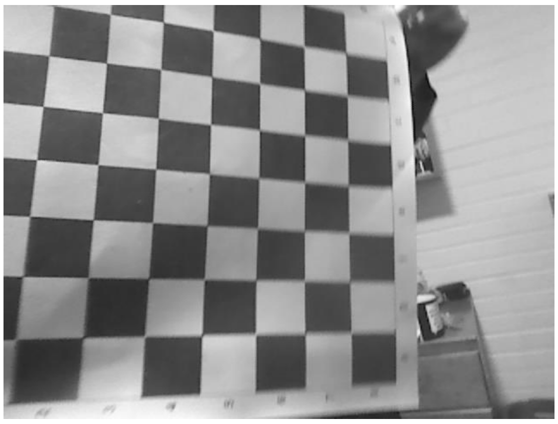
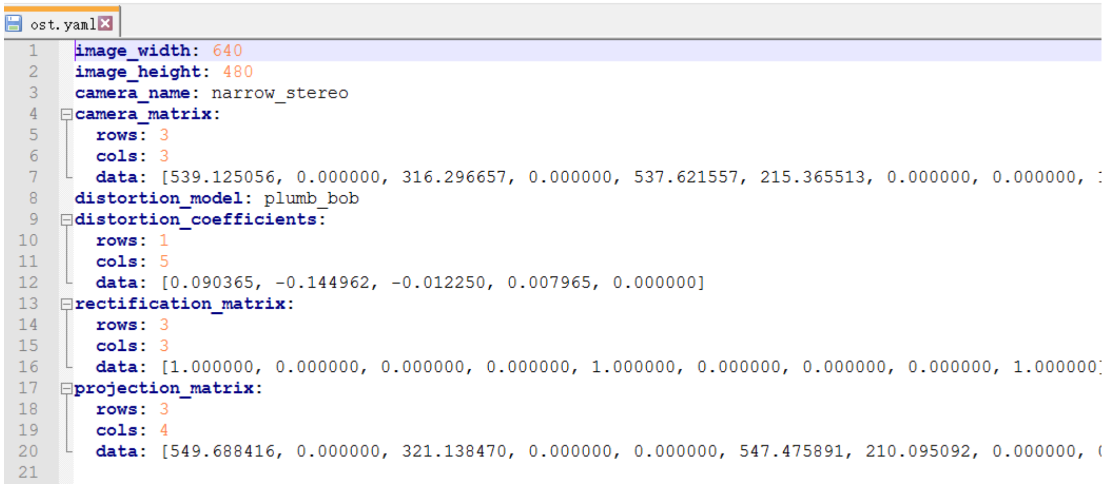


Figure 2. 14 An example saved calibration picture



ost.yaml

```
image_width: 640
image_height: 480
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [539.125056, 0.000000, 316.296657, 0.000000, 537.621557, 215.365513, 0.000000, 0.000000, 
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.090365, -0.144962, -0.012250, 0.007965, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 1.000000
projection_matrix:
  rows: 3
  cols: 4
  data: [549.688416, 0.000000, 321.138470, 0.000000, 0.000000, 547.475891, 210.095092, 0.000000, 
```

Figure 2. 15 The calibration result, the camera rgb info

3. Process and result

After introducing all the backgrounds used in this project, now the project can start. This chapter are mainly about how to finish all the desired functions from the developing environment configuration, tracking functions satisfaction to testing results. All the steps are shown below clearly with necessary description for better understanding of functions, algorithm and the meaning of each step.

3.1 Config developing environment

3.1.1 Install ROS kinetic in ubuntu 16.04

First, it is necessary to update the ubuntu packages to ensure there are all ROS packages available to install

```
sudo apt-get update
```

Then, the installation of ROS can start. Input the command in the terminal to install full version of kinetic

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

This command needs the authority of users, user need to do authorization by input the password. And after all packages are finished, it is also necessary to initialize the dependence and activate the ROS according to add the setup.bash path to environment path ~/.bashrc which is

```
sudo rosdep init
rosdep update
echo “source /opt/ros/kinetic/setup.bash” >> ~/.bashrc
```

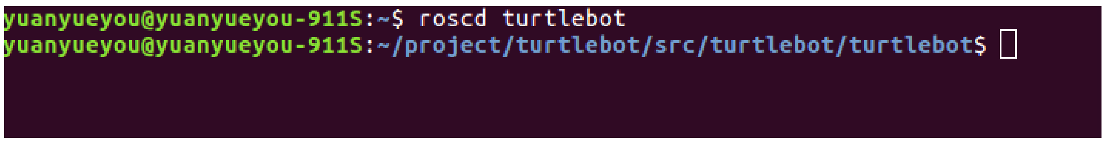
the ‘echo’ command above can add the source path into .bashrc file, where the bashrc file is an environment path, each time a new terminal is opened, the path in bashrc file will reload once to setup the developing environment. So here, adding the ROS setup file into the path file to ensure every time the terminal is opened, the developing environment is in ROS. And then install some important dependence below.

sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential

By now, all the necessary packages are installed and it is time to check whether the kinetic ROS has installed successfully. Input the roscore to start the master in ROS, then rosrun turtlesim turtlesim_node, if there is a turtle logo shown on the screen, it shows that ROS has installed successfully.

3.1.2 Install turtlebot model

There is a very clear introduction about how to install turtlebot model in the turtlebot wiki website http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot%20Installation. Everyone can follow these steps shown on this website to download their own turtlebot models onto their ROS environment. Here list some steps of install turtlebot model. User need to build three workspace which named rocon, kobuki and turtlebot, and then download the corresponding git code and the do the cmake. Besides, add these three workspaces setup file path to the bashrc file. Each time a new terminal is opened, three workspaces are setup. To check if the turtlebot has been installed successfully, just input command 'roscd turtlebot' in the terminal. If the terminal can locate turtlebot package and change directory to turtlebot/devel, then the turtlebot is installed smoothly. Here is an example of checking the installation of turtlebot shown in Figure 3.1



```
yuanyueyou@yuanyueyou-911S:~$ roscd turtlebot
yuanyueyou@yuanyueyou-911S:~/project/turtlebot/src/turtlebot/turtlebot$ 
```

Figure 3. 1 After input roscd turtlebot, the terminal can change directory to turtlebot

3.1.3 Install the newest version of opencv

As mentioned at background, it is necessary to update the verion of opencv, the version after 3.4.2 start to support the YOLO algorithm. In ROS kinetic, opencv-3.3.1-dev is installed by default, and the python environment of ROS is only version

2.7 by default. After installing other versions of OpenCV through pip, ROS will call the default 3.3.1 first, so it is important to uninstall this version first:

```
sudo easy_install trash-cli
sudo trash-put /opt/ros/kinetic/lib/python2.7/dist-packages/cv2.so
```

To install a trash cli plug-in, it is convenient to directly delete cv2.so under the ROS installation directory. Then it is available to use the pip command to install any version of OpenCV and its extension package even the newest opencv version. By now, the newest version of opencv-python is 4.2.0.34, users can download this opencv by using pip command. for more information, please check the offical opencv-python website https://pypi.org/project/opencv-python/. There are all the version of opencv and users can download any version of opencv here.

```
pip install opencv-python
pip install opencv-contrib-python
```

And if users want to uninstall the opencv, just change ‘install’ to ‘uninstall’ for above commands, then the opencv can be uninstalled successfully.

3.2 Build the simulation Wrold

3.2.1 Build the world

It is convenient to do the simulation in the simulation world, and gazebo is a good virtual world software. By using gazebo, users can build their own virtual world following they desire. And there are many models such like the walking person, walls, ambulance and so on in gazebo which greatly convenient users to do their creation. In this project, for people detection and people re-id, two different people models are needed. Gazebo has provided the standing person and walking person models, but these two types person are the same with white t-shirt and blue jeans. So there is one more people model waited for the people recognition and re-id technologies.

For another person which are not included in the gazebo library, it is necessary to build a new person or just download from the models website.

https://3dwarehouse.sketchup.com/search/?q=person&login=true

Download the dae file. For this project, a brown_person model is download from the models website, there is a model.dae file in the compressed zip file. Then add two files named model.config and model.sdf in the brown_person folder. The content of these two new created file can be copied from the walking_person floder which locates ~/.gazebo/models. After copying these two files model.config and model.sdf to the brown_person folder, just to change the mesh file path to the brown_person model.dae file in model.sdf. And also change the model name in model.config to brown_person. Finally, move the folder into ~/.gazebo/models. By now, when the gazebo is opened again, there is a brown_person model available in the insert section and user can use this model download from website. And Figure 3.2 is the final simulation world this project applies.

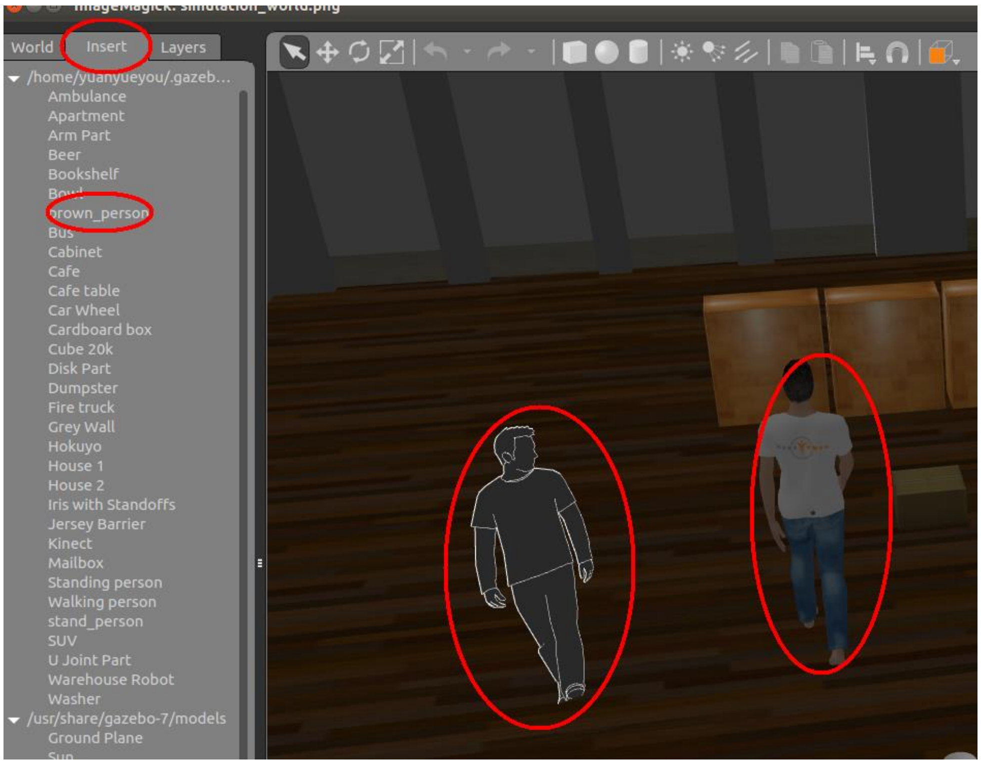


Figure 3. 2The simulation world with download brown_person and the walking person, the left person is the brown person model, the right is walking person available in gazebo library

3.2.2 Move the models in gazebo

To test the tracking task in the simulation world, it is important to make the people movable and the robot can track the people.

For the procedure about how to make the model movable in gazebo world, please check the ROS ignite website https://www.robotigniteacademy.com/en/ and the URDF and TF courses in ros for beginners

3.3 YOLO detection

In this section, the YOLO algorithm is used to do the object detection. This YOLO algorithm will return the detection result which includes the object location and the width and height of the detected objects. However, this algorithm will return the ratio values between 0 and 1. So if users want to know what is the location in pixels, then the first step is using ratio values multiple the width or height of the window.

Besides, it is also necessary to download the weights file and the corresponding config file. Due to this project applying the yolov3, so download yolov3.weights and yolov3.cfg file from the offical website. And do not forget also download the coco.name, of course users can train their own class and add it into coco.name file.

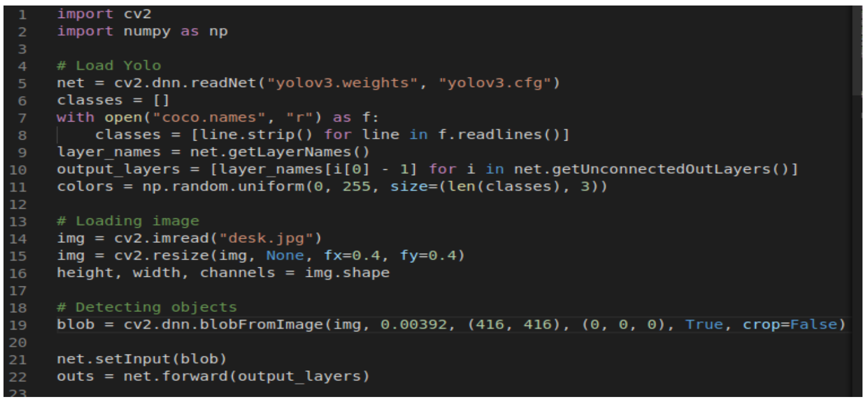
Then, write some code to test whether the yolo algorithm is useful. The code is shown in Figure 3.3. All the project code is written by python.

First, yolo detection need to deal with the image problem, so here import cv2, which is import the opencv functions into these python script. For more powerful matrix handling system, importing numpy is also necessary.

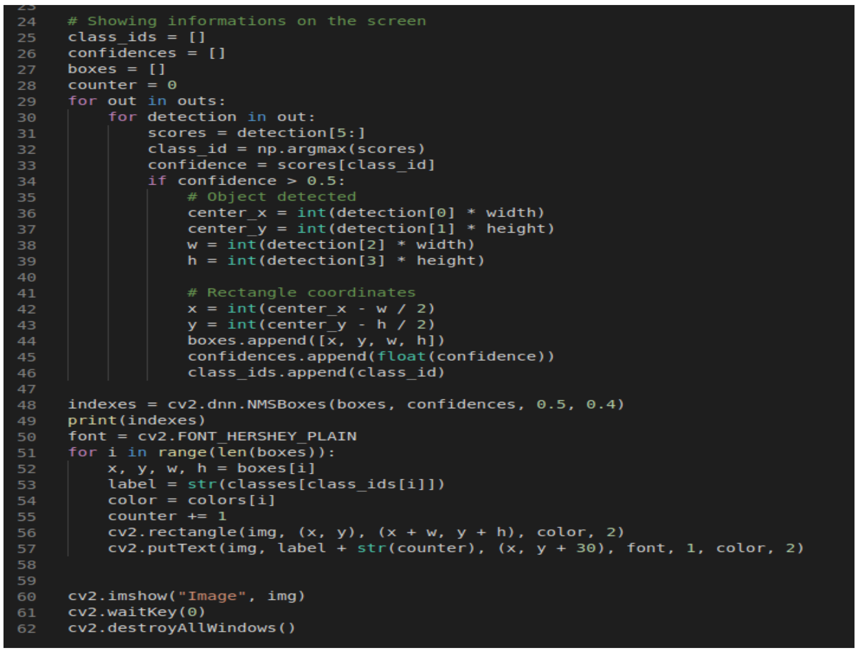
Then load the yolo algorithm by using the readNet function shown at line 5. This function is used to build a Deep Neural Network (for continuing contents this network is called dnn) by reading the input weights file and cfg file, besides, this function is only supported after opencv version 3.4.2, and that is why the project need to download the newest version of opencv, of course the newest the version support the newest functions. Then name the built dnn network as ‘net’ for continuing usage.

Load all the class saved in the coco.name file from line to line and give these class names to classes[] matrix. Then traversal all the layer_names to get the output layers where the output layers save the detection results of yolo.

Next, input the image to the dnn network to do the detection shown on # Load image. And get the width and height of the image by using cv2. This desk.jpg is the test image of yolo detection.



```python
import cv2
import numpy as np

# Load Yolo
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))

# Loading image
img = cv2.imread("desk.jpg")
img = cv2.resize(img, None, fx=0.4, fy=0.4)
height, width, channels = img.shape

# Detecting objects
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)
outs = net.forward(output_layers)
```

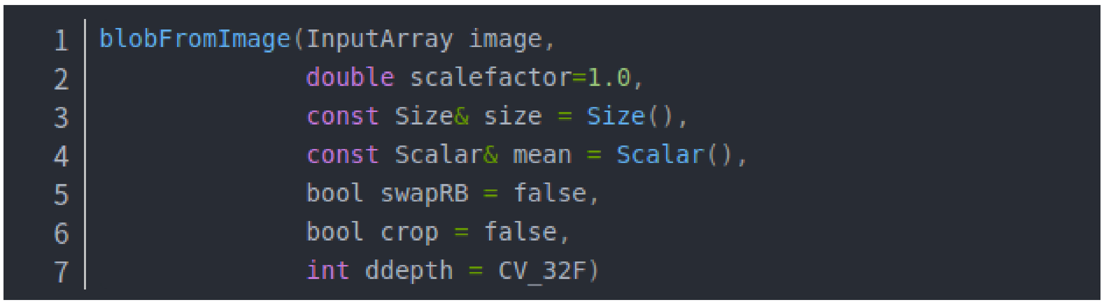


```python
# Showing informations on the screen
class_ids = []
confidences = []
boxes = []
counter = 0
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            # Object detected
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)
            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
print(indexes)
font = cv2.FONT_HERSHEY_PLAIN
for i in range(len(boxes)):
    x, y, w, h = boxes[i]
    label = str(classes[class_ids[i]])
    color = colors[i]
    counter += 1
    cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
    cv2.putText(img, label + str(counter), (x, y + 30), font, 1, color, 2)


cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 3. 3 the code of testing YOLO detection

And by now, all the preparations are finished, the detection can start. The blobFromImage function is used for transfer the image to blobs because the dnn network set the blob type of data as the input.
Here it is essential to introduce the usage of this function,In deep learning or image classification, blobfromimage is mainly used to pretreatment images. There are two main processes:

Overall pixel value minus mean and scale the image pixel value by scale factor.
Figure 3.4 shows the structure of this function and the input value types.



```
blobFromImage(InputArray image,
              double scalefactor=1.0,
              const Size& size = Size(),
              const Scalar& mean = Scalar(),
              bool swapRB = false,
              bool crop = false,
              int ddepth = CV_32F)
```

Figure 3. 4 The structure of blobFromImage function

For the first parameter, image is the input image that is used to do the detection or classification in dnn network.
Mean: mean is the average value of the whole picture pixels need to subtract. If there is need to subtract different values from the three channels of the RGB picture, three groups of average values can be used. If only one group value is used, the system will subtract the same value from the three channels by default which is mean.
Scalefactor: after subtracting the average value from the picture, the remaining pixel values can be scaled to a certain extent. Its default value is 1. However, in the yolo detection the saclefactor is 0.00392 which can be found here
https://github.com/opencv/opencv/blob/master/samples/dnn/models.yml

Size: this parameter is the image size that the neural network requires to input when training. And for yolo detection, there are three size available, 320*320, 416*416 and 608*608 as mentioned at background.

Swaprb: in opencv, the picture channel order is BGR, but the image encoding generally is RGB, so if there is needs to exchange R and G, make sure swaprb = true.

And then input the blobs into the dnn network, the detection result can be got. After forwarding these detection results to the output layers, the final outputs come out. Here the detection results are saved in outs matrix.

From line 24 to the end, the code mainly finishes the displaying of detections whose confidences are greater than 0.5 with bounding boxes and the class names. And Figure 3.5 shows the test results of the yolo detections.

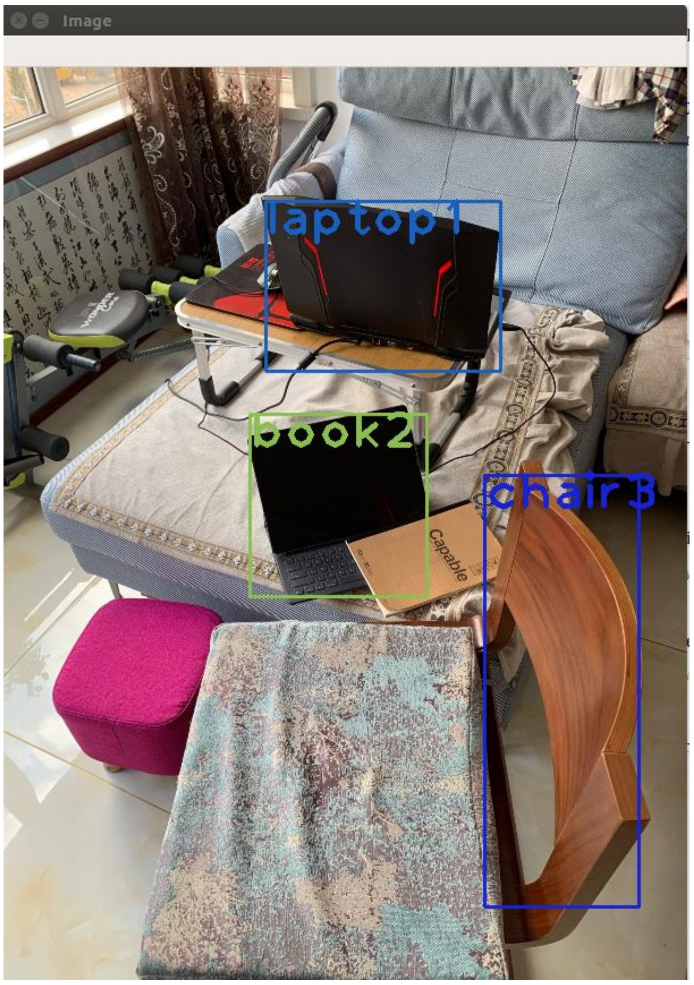


Figure 3. 5 The testing result of yolov3 detection

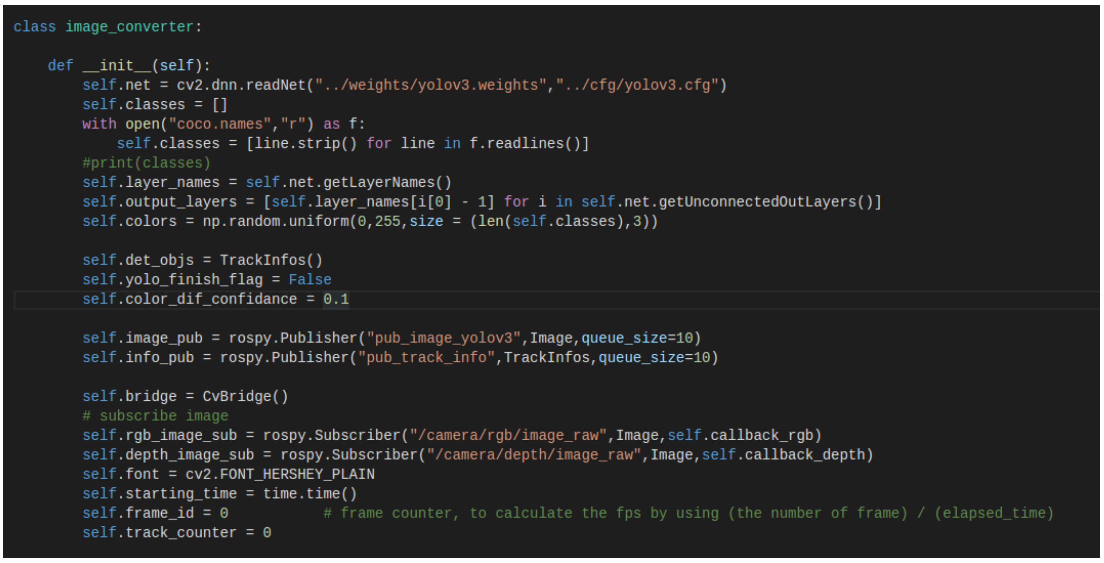
3.4 Image converter

3.4.1 image converter introduction

In this section, for more convenient usage of the yolo detection and the image processing, a class named image_converter is built. This image_converter class is the base of the whole project detection part, and the continuing parts Re-id and getting distance are also the new functions with the basic of image_converter class. And all the codes for detection part is written in the real_time_yolo.py

3.4.2 define image_converter class

At the initialization step, the class define the member variables include the basic steps of yolo detection which are similar to the test code of yolo detection. And define two subscribers which subscribe the camera rgb information and the camera depth information. By listening these two topics and receiving image data, do the yolo detection and get the distance in the callback functions. Besides there is also a publisher as the member variable to publish the detected objects information which is a self-defined message type TrackInfos and TrackInfo through "pub_track_info" topic. These two message types are described detail in the next section. At the move base part, the move_base subscribes the "pub_track_info" topic, and according to received track information, move_base starts to move forward or drawback to satisfy the tracking function. Figure 3.6 shows the __init__ function of this image_converter class.



```python
class image_converter:

    def __init__(self):
        self.net = cv2.dnn.readNet("../weights/yolov3.weights","../cfg/yolov3.cfg")
        self.classes = []
        with open("coco.names","r") as f:
            self.classes = [line.strip() for line in f.readlines()]
        #print(classes)
        self.layer_names = self.net.getLayerNames()
        self.output_layers = [self.layer_names[i[0] - 1] for i in self.net.getUnconnectedOutLayers()]
        self.colors = np.random.uniform(0,255,size = (len(self.classes),3))

        self.det_objs = TrackInfos()
        self.yolo_finish_flag = False
        self.color_dif_confidance = 0.1

        self.image_pub = rospy.Publisher("pub_image_yolov3",Image,queue_size=10)
        self.info_pub = rospy.Publisher("pub_track_info",TrackInfos,queue_size=10)

        self.bridge = CvBridge()
        # subscribe image
        self.rgb_image_sub = rospy.Subscriber("/camera/rgb/image_raw",Image,self.callback_rgb)
        self.depth_image_sub = rospy.Subscriber("/camera/depth/image_raw",Image,self.callback_depth)
        self.font = cv2.FONT_HERSHEY_PLAIN
        self.starting_time = time.time()
        self.frame_id = 0           # frame counter, to calculate the fps by using (the number of frame) / (elapsed_time)
        self.track_counter = 0
```

Figure 3. 6 The __init__ function of image_converter

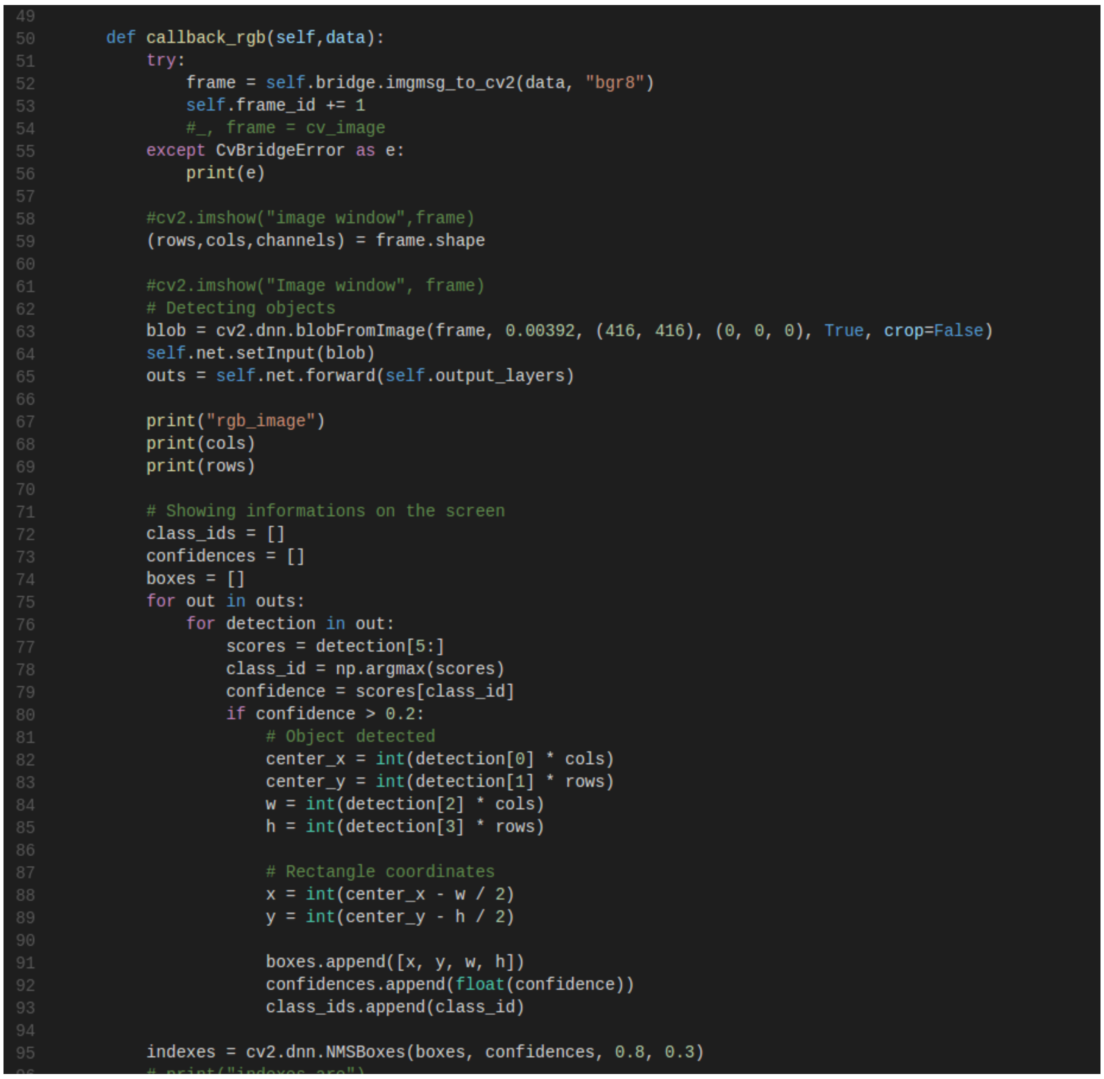
Besides, there are some variables which include the font used in showing on the screen, the start_time which is used to calculate the FPS. There is a very important member variable which is self.brdige = CvBridge().

CvBridge is a ROS library which can connect the sensor_msgs/Image information and the opencv together. CvBridge can transfer the received Image msg to the desired image data with suitable encoding. So here the CvBridge will be used to transfer the received color messages and depth messages to opencv format. Because the opencv format data are much convenient to process and do the detection.

As mentioned before, there are two subscribers, and each subscriber has their own callback function, the one is called callback_rgb(), the other called callback_depth(). The callback_rgb() function mainly deal with the object detection and the re-id function, and the callback_depth() function mainly get the distance between the detected objects and the camera. Figure 3.7 shows the object detection code in callback_rgb().

The input data parameter of callback_rgb function is the received data flow from the topic /camera/rgb/image_raw, then the CvBridge is used to transfer the Image format data to the desired ‘bgr8’ format data, the bgr8 format data is one of the most popular type, there are so many functions service for this type of data. The global variable frame_id is used to calculate the total number of frames from the starting of received Image msg data. And the frame_id divide the elapsed time, the FPS(frames per second) will be got. FPS = total frame number/ elapsed time.

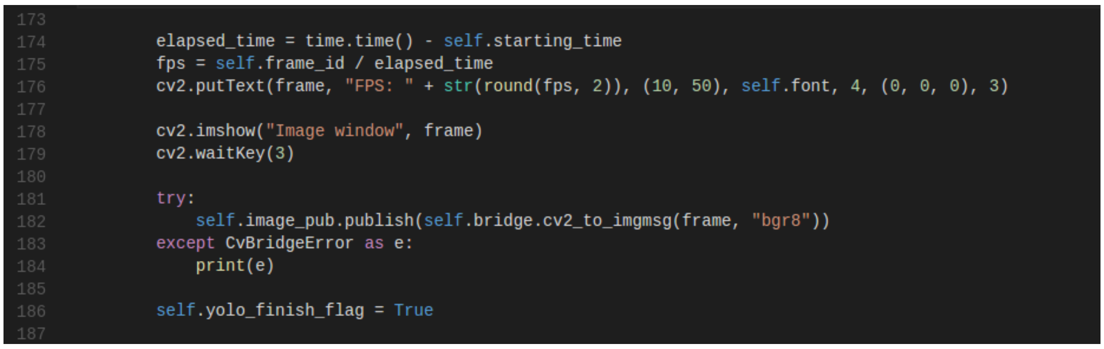
Then, the yolo detection starts to work and save the detected object information including coordination, width, heights into boxes[] matrix and save the confidence information into confidences[] matrix.



```python

    def callback_rgb(self,data):
        try:
            frame = self.bridge.imgmsg_to_cv2(data, "bgr8")
            self.frame_id += 1
            #_, frame = cv_image
        except CvBridgeError as e:
            print(e)

        #cv2.imshow("image window",frame)
        (rows,cols,channels) = frame.shape

        #cv2.imshow("Image window", frame)
        # Detecting objects
        blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
        self.net.setInput(blob)
        outs = self.net.forward(self.output_layers)

        print("rgb_image")
        print(cols)
        print(rows)

        # Showing informations on the screen
        class_ids = []
        confidences = []
        boxes = []
        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > 0.2:
                    # Object detected
                    center_x = int(detection[0] * cols)
                    center_y = int(detection[1] * rows)
                    w = int(detection[2] * cols)
                    h = int(detection[3] * rows)

                    # Rectangle coordinates
                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)

                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

        indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.8, 0.3)
```

Figure 3. 7 The object detection part of the callback_rgb() function

Besides, Figure 3.8 shows the end code of the callback_rgb() which describe how to calculate the FPS and show the FPS data on the screen. Notice that here is a member variable called yolo_finish_flag, whose default value is False defined at the __init__ function. This flag is used to make callback_rgb() and callback_depth() synchronized. Due to the different calculation of these two callback functions, the running time is not the same for one period and generally the callback_depth() run much shorter time than callback_rgb(). So, the yolo_finish_flag is defined and the default value is False. The yolo_finish_flag will not turn to True until all the yolo detections are finished in callback_rgb(). At the same time, there is a while loop in callback_depth(), wait until the flag turns to True which means the callback_rgb() finishes all the calculation for one period. And by using this method, the project makes it possible to synchronize these two callback functions.

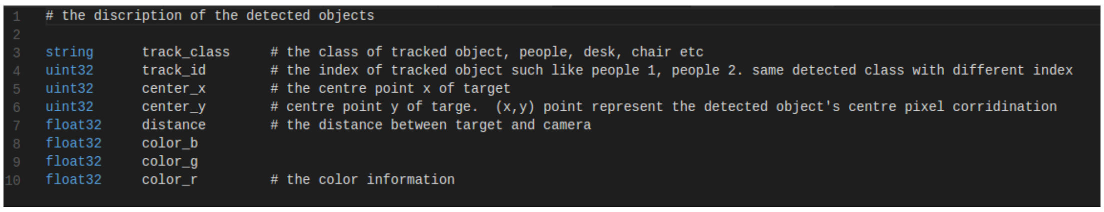


```
        elapsed_time = time.time() - self.starting_time
        fps = self.frame_id / elapsed_time
        cv2.putText(frame, "FPS: " + str(round(fps, 2)), (10, 50), self.font, 4, (0, 0, 0), 3)

        cv2.imshow("Image window", frame)
        cv2.waitKey(3)

        try:
            self.image_pub.publish(self.bridge.cv2_to_imgmsg(frame, "bgr8"))
        except CvBridgeError as e:
            print(e)

        self.yolo_finish_flag = True
```

Figure 3. 8 FPS calculating and showing on screen in real time

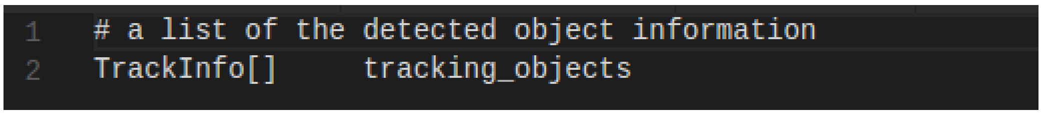
3.4.3 create TrackInfos and TrackInfo msg

As mentioned in 3.3.2, the image_converter class will publish the detected objects information. And the move base will move around according to these information. So there are two self defined messages used in this project TrackInfo.msg and TrackInfos.msg. For the TrackInfo msg, it describe the characteristics of the detected objects which include the class name, the track index to identify each objects, the coordination with pixel unit, the distance between the target and camera and the color information. For the TrackInfos msg, this messages is a list of TrackInfo objects, and the self.info_pub publisher will publish the TrackInfos messages to ‘pub_track_info’ topic. Figure 3.9 and Figure 3.10 show the definition of TrackInfo.msg and TrackInfos.msg respectively.



```
# the discription of the detected objects

string      track_class    # the class of tracked object, people, desk, chair etc
uint32      track_id       # the index of tracked object such like people 1, people 2. same detected class with different index
uint32      center_x       # the centre point x of target
uint32      center_y       # centre point y of targe.  (x,y) point represent the detected object's centre pixel corridination
float32     distance       # the distance between target and camera
float32     color_b
float32     color_g
float32     color_r        # the color information
```

Figure 3. 9 The definition of TrackInfo message

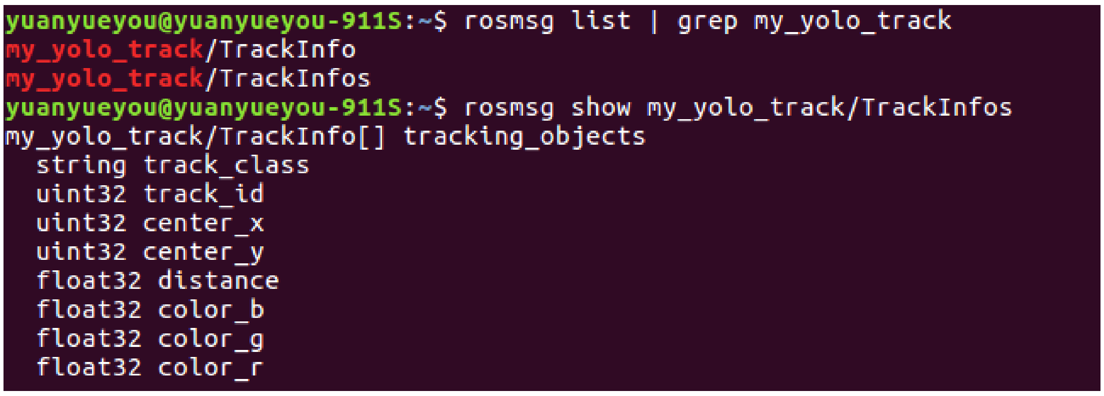


```
# a list of the detected object information
TrackInfo[]     tracking_objects
```

Figure 3. 10 The definition of TrackInfos message

After defining these two types of message, change the CMakeList.txt and package.xml in the my_yolo_track package to make sure this project can use these two self-defined messages.

After changing these two files, return the workspace and catkin_make again. Finally, there is messages available for this project. Use the command rosmsg show my_yolo_track/TrackInfo to check. Figure 3.11 is the result after successful catkin_make.



```
yuanyueyou@yuanyueyou-911S:~$ rosmsg list | grep my_yolo_track
my_yolo_track/TrackInfo
my_yolo_track/TrackInfos
yuanyueyou@yuanyueyou-911S:~$ rosmsg show my_yolo_track/TrackInfos
my_yolo_track/TrackInfo[] tracking_objects
  string track_class
  uint32 track_id
  uint32 center_x
  uint32 center_y
  float32 distance
  float32 color_b
  float32 color_g
  float32 color_r
```

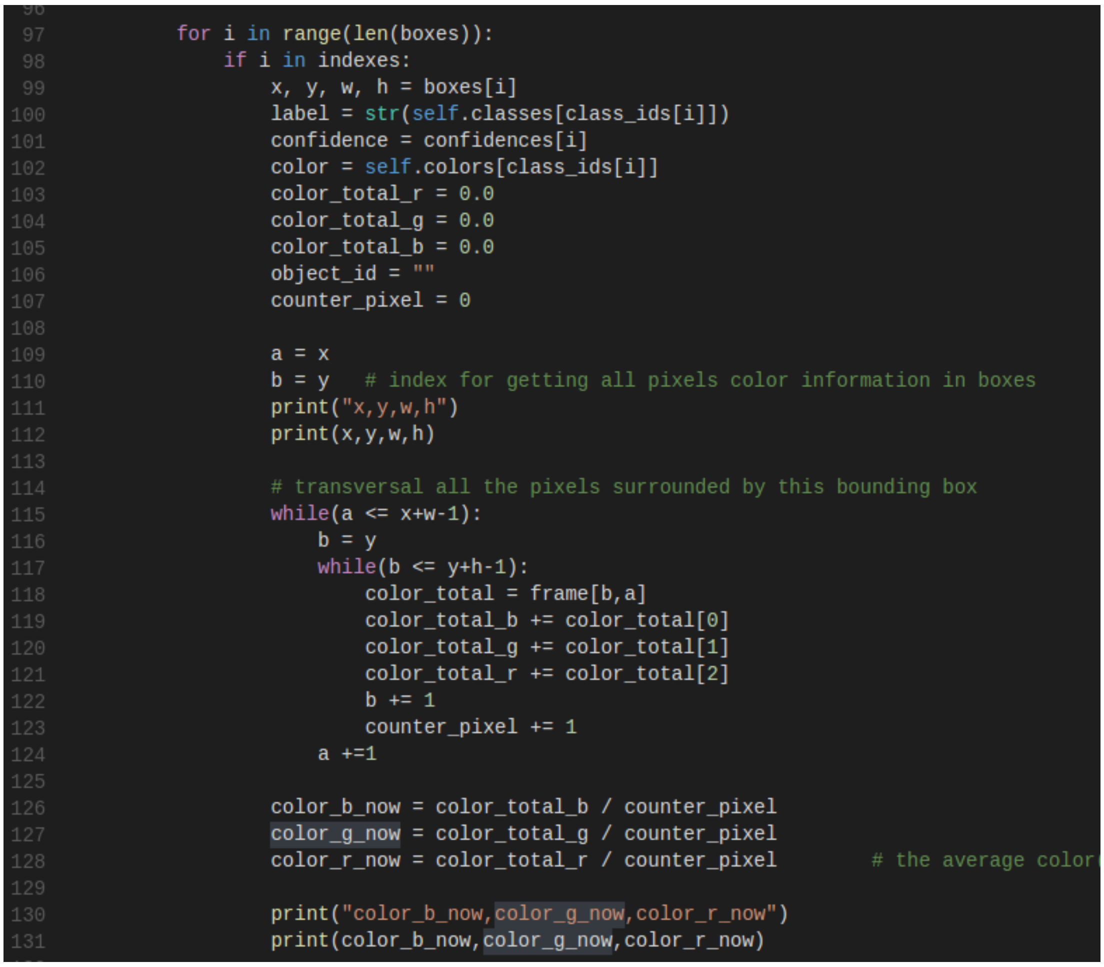
Figure 3. 11 The self-defined message information of TrackInfos

3.5 Re-id

As mentioned at 3.4, the object detection information such like coordination, width and height are saved in boxes[] matrix. Here the re-id function continue the code of yolo detection in callback_rgb().

For each detected object, to identify different objects, it is necessary to define and save the characteristic of each object. Here the color information is selected as this identification character.　Every visible object has its own color information, if two same class objects from two frames have similar color information, there is high percentage that the two objects are the same one. Hence this project applies this method to do re-id.

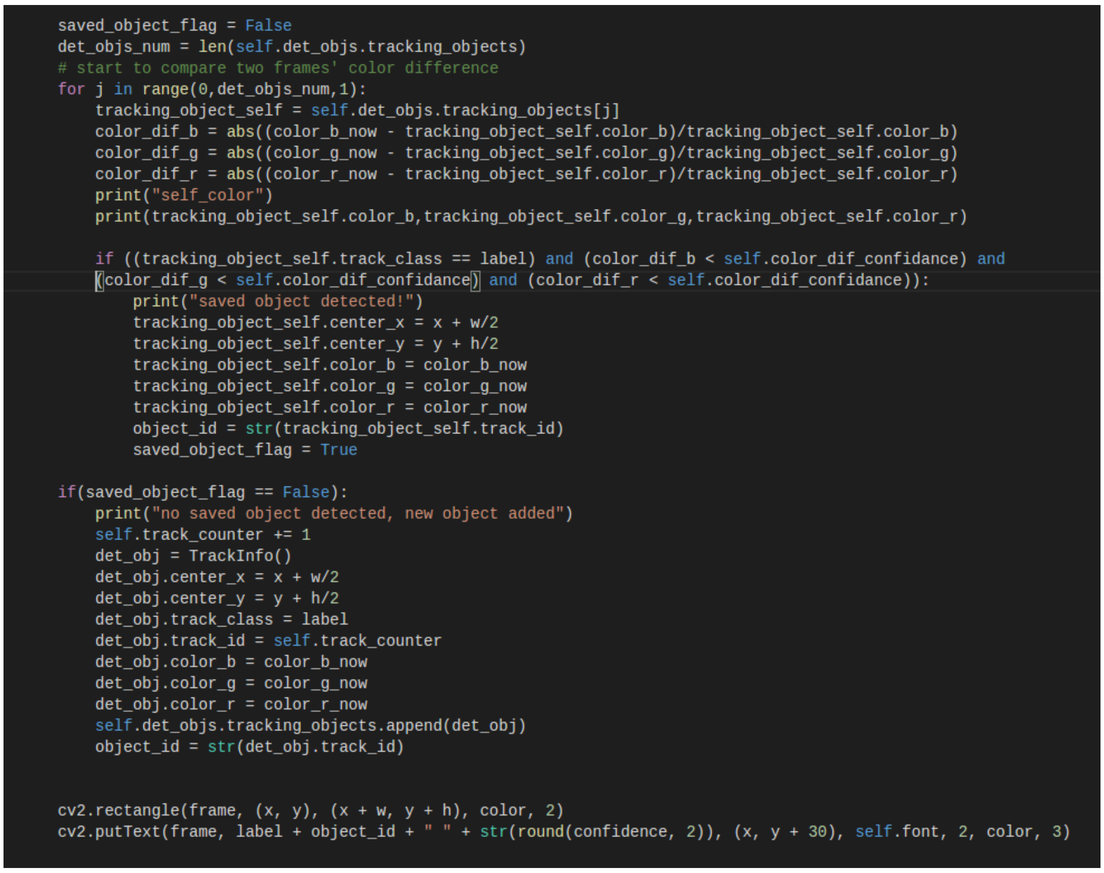
First, define a member variable self.det_objs = TrackInfos (). This matrix saves the detected object information and the total information is shown in Figure 3.11. For each detected object, their coordination and size are saved in boxes []. So, the number of boxes is the same as the number of detected objects. For each bounding box, the project starts to traversal all the pixels included in the bounding box, and then add each pixel color information together. For each pixel, the bgr8 format image is a 3 channels data format, the first channel contains the color_red information, the second contains the color_green information and the third channel contains color_blue information. All the color information adds together to get the total color number, here defined color_total_r, color_total_g and color_total_b. Then the total color number divide the all number of pixels included in the bounding box, then there is average color information for each primary color which are red, green and blue. And set this group of average color information as the object color information and save this information to the global self.det_objs matrix variable. The Figure 3.12 shows the codes how to calculate the color information for re-identification.



```
        for i in range(len(boxes)):
            if i in indexes:
                x, y, w, h = boxes[i]
                label = str(self.classes[class_ids[i]])
                confidence = confidences[i]
                color = self.colors[class_ids[i]]
                color_total_r = 0.0
                color_total_g = 0.0
                color_total_b = 0.0
                object_id = ""
                counter_pixel = 0

                a = x
                b = y   # index for getting all pixels color information in boxes
                print("x,y,w,h")
                print(x,y,w,h)

                # transversal all the pixels surrounded by this bounding box
                while(a <= x+w-1):
                    b = y
                    while(b <= y+h-1):
                        color_total = frame[b,a]
                        color_total_b += color_total[0]
                        color_total_g += color_total[1]
                        color_total_r += color_total[2]
                        b += 1
                        counter_pixel += 1
                    a +=1

                color_b_now = color_total_b / counter_pixel
                color_g_now = color_total_g / counter_pixel
                color_r_now = color_total_r / counter_pixel        # the average color

                print("color_b_now,color_g_now,color_r_now")
                print(color_b_now,color_g_now,color_r_now)
```

Figure 3. 12The code of calculating the color information

Then the project start to compare the current bounding box average color information and the objects’ color information saved in self.det_objs matrix which are shown in Figure 3.13. Here is a save_object_flag is defined and the initial value is False, this flag is to judge whether the current bounding box average color information(this name is called CBBAC in the following content) has a very high similarity with someone object color information in self.det_objs matrix. If the current detected object has the same class name and the color difference which is lower than a color difference confidence, then this current detected object is seen as the same as some object saved in objects matrix which is self.det_objs matrix and the saved_object_flag is set True at the same time. Besides, the saved object color information and coordination will update according to save the current detected object's color information and coordinate.



```
saved_object_flag = False
det_objs_num = len(self.det_objs.tracking_objects)
# start to compare two frames' color difference
for j in range(0,det_objs_num,1):
    tracking_object_self = self.det_objs.tracking_objects[j]
    color_dif_b = abs((color_b_now - tracking_object_self.color_b)/tracking_object_self.color_b)
    color_dif_g = abs((color_g_now - tracking_object_self.color_g)/tracking_object_self.color_g)
    color_dif_r = abs((color_r_now - tracking_object_self.color_r)/tracking_object_self.color_r)
    print("self_color")
    print(tracking_object_self.color_b,tracking_object_self.color_g,tracking_object_self.color_r)

    if ((tracking_object_self.track_class == label) and (color_dif_b < self.color_dif_confidance) and
    (color_dif_g < self.color_dif_confidance) and (color_dif_r < self.color_dif_confidance)):
        print("saved object detected!")
        tracking_object_self.center_x = x + w/2
        tracking_object_self.center_y = y + h/2
        tracking_object_self.color_b = color_b_now
        tracking_object_self.color_g = color_g_now
        tracking_object_self.color_r = color_r_now
        object_id = str(tracking_object_self.track_id)
        saved_object_flag = True

if(saved_object_flag == False):
    print("no saved object detected, new object added")
    self.track_counter += 1
    det_obj = TrackInfo()
    det_obj.center_x = x + w/2
    det_obj.center_y = y + h/2
    det_obj.track_class = label
    det_obj.track_id = self.track_counter
    det_obj.color_b = color_b_now
    det_obj.color_g = color_g_now
    det_obj.color_r = color_r_now
    self.det_objs.tracking_objects.append(det_obj)
    object_id = str(det_obj.track_id)


cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
cv2.putText(frame, label + object_id + " " + str(round(confidence, 2)), (x, y + 30), self.font, 2, color, 3)
```

Figure 3. 13 The code of comparing the current object information with the saved detected objects information for re-identification

However, if there is no saved objects have quite high similarity with the current detected object, a new object with the coordination, class name, color information and the track index will be appended in the objects matrix. And finally show the necessary information on the screen. The whole process flow chart can be seen at Figure 3.14 and the final re-id result can be seen at figure 3.15.

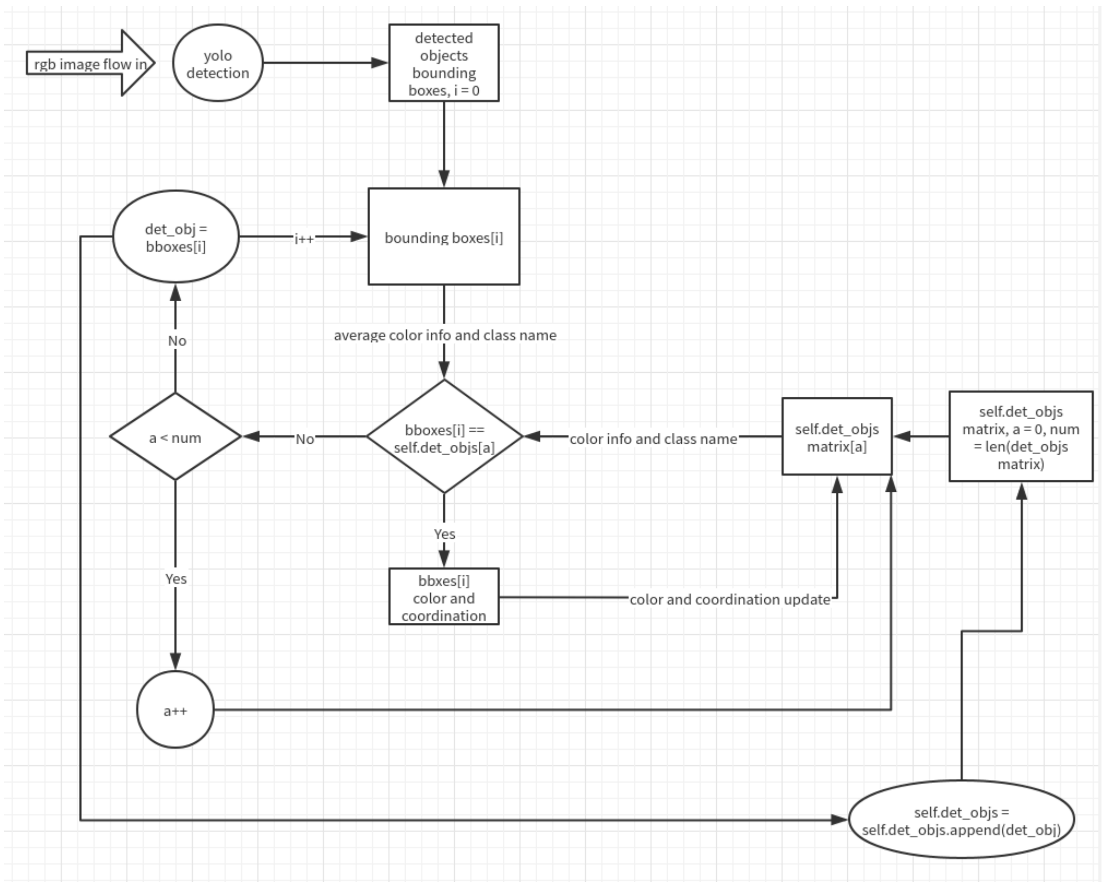

rgb image flow in
yolo detection
detected objects bounding boxes, i = 0
det_obj = bboxes[i]
i++
bounding boxes[i]
average color info and class name
No
a < num
No
bboxes[i] == self.det_objs[a]
color info and class name
self.det_objs matrix[a]
self.det_objs matrix, a = 0, num = len(det_objs matrix)
Yes
Yes
bbxes[i] color and coordination
color and coordination update
a++
self.det_objs = self.det_objs.append(det_obj)

Figure 3. 14 The logic flow chart of re-id function

As shown in Figure 3.13 the two different person with different index and the confidence shown on the screen, and at the same time, the saved detected objects information also is shown in the terminal. The class name, track id, color information and coordination with center_x and center_y can be obtained from callback_rgb() function. The distance is obtained at callback_depth() function which will be describe in next section.

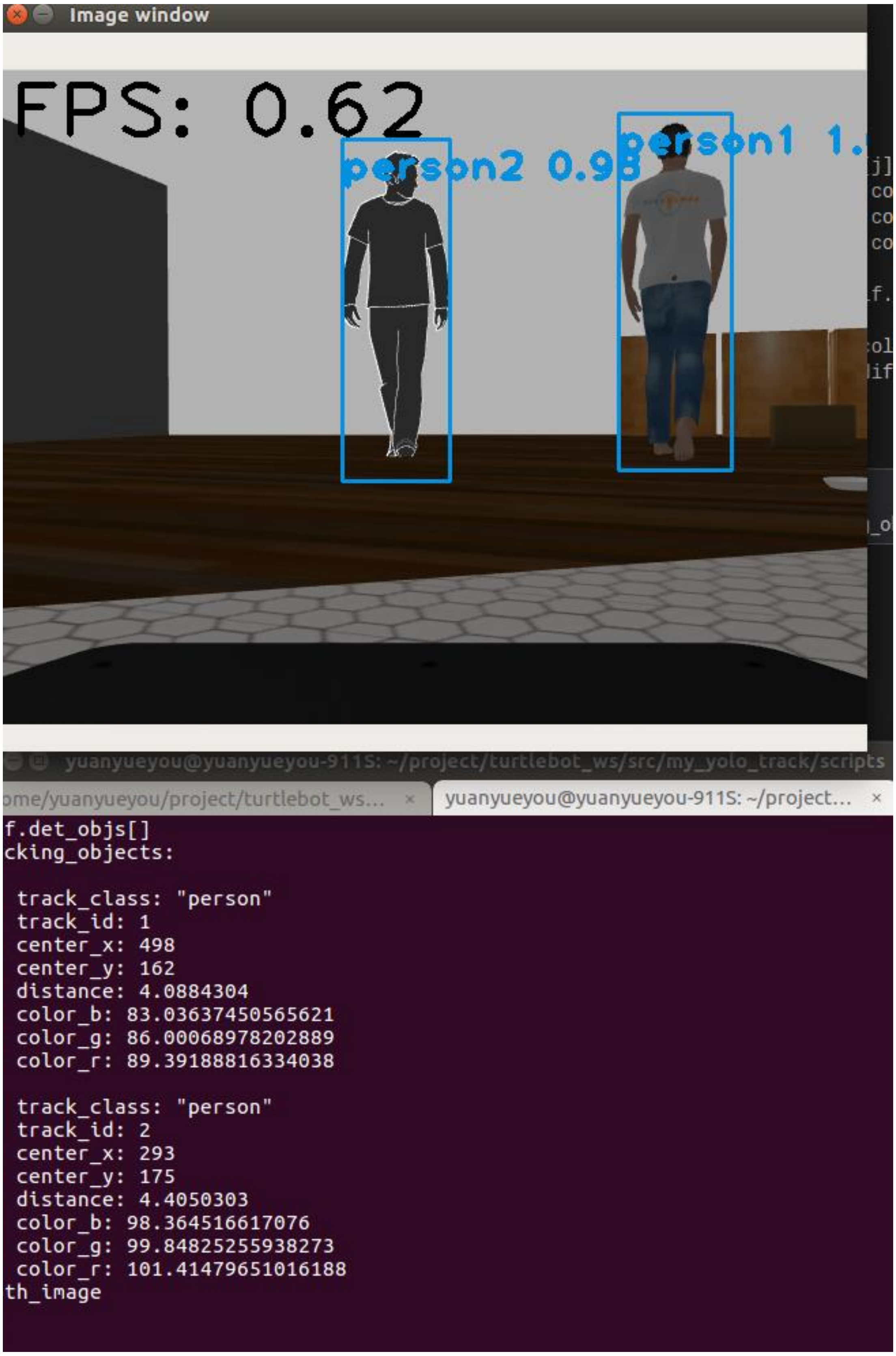


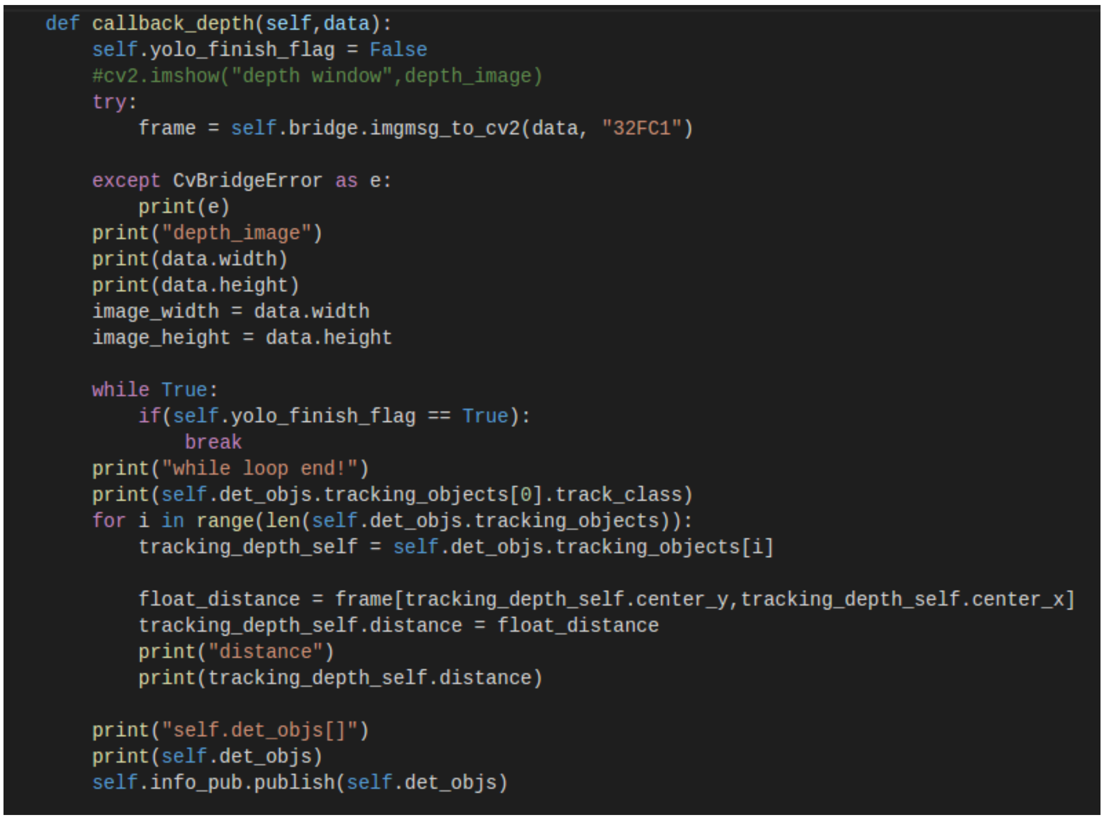
Image window
FPS: 0.62
person2 0.98
person1 1.
yuanyueyou@yuanyueyou-911S: ~/project/turtlebot_ws/src/my_yolo_track/scripts
ome/yuanyueyou/project/turtlebot_ws...
yuanyueyou@yuanyueyou-911S: ~/project...

```
f.det_objs[]
cking_objects:

 track_class: "person"
 track_id: 1
 center_x: 498
 center_y: 162
 distance: 4.0884304
 color_b: 83.03637450565621
 color_g: 86.00068978202889
 color_r: 89.39188816334038

 track_class: "person"
 track_id: 2
 center_x: 293
 center_y: 175
 distance: 4.4050303
 color_b: 98.364516617076
 color_g: 99.84825255938273
 color_r: 101.41479651016188
th_image
```

Figure 3. 15 The re-id results

3.6 Get distance from target to robot

This section mainly describe how to get the distance between the detected objects and camera by using the depth information in callback_depth(). And Figure 3.16 shows the code of how deal with the depth message, and use the depth information for the distance.

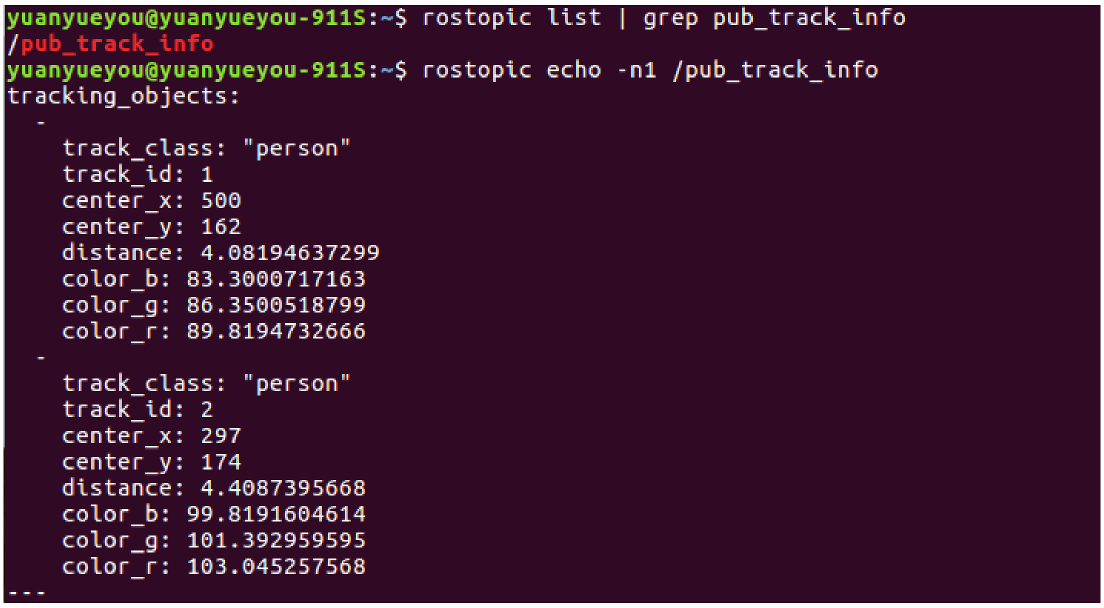


```python
def callback_depth(self,data):
    self.yolo_finish_flag = False
    #cv2.imshow("depth window",depth_image)
    try:
        frame = self.bridge.imgmsg_to_cv2(data, "32FC1")

    except CvBridgeError as e:
        print(e)
    print("depth_image")
    print(data.width)
    print(data.height)
    image_width = data.width
    image_height = data.height

    while True:
        if(self.yolo_finish_flag == True):
            break
    print("while loop end!")
    print(self.det_objs.tracking_objects[0].track_class)
    for i in range(len(self.det_objs.tracking_objects)):
        tracking_depth_self = self.det_objs.tracking_objects[i]

        float_distance = frame[tracking_depth_self.center_y,tracking_depth_self.center_x]
        tracking_depth_self.distance = float_distance
        print("distance")
        print(tracking_depth_self.distance)

    print("self.det_objs[]")
    print(self.det_objs)
    self.info_pub.publish(self.det_objs)
```

Figure 3. 16The code of how to get the distance between detected objects and camera

The callback_depth() subscribe the depth information through "camera/depth/image_raw" topic, and the yolo_finish_flag is used to synchronize callback_rgb() and callback_depth(). Then the opencv try to convert the depth msg format to the 32FC1 format which is a depth information format. And after the message type transfer, the frame contains the distance information in meter for each pixel. Hence, this callback_depth() starts to traversal all the detected objects saved in self.det_objs matrix, and get the coordination of each object which is the center x and y of the detected object with pixel unit. And then according to the coordination the distance will be get by using distance = frame[rows,cols] = frame[y,x]. Because for the image, the pixel is located by the rows and columns which corresponding to the y and x respectively. After get the distance, then update the distance information by saving the detected distance to self.det_objs.tracking_objs[i].distance. And finally all the information the objects needed are obtained. And the final testing result is shown in Figure 3.15, there is distance available.

Then the last step is publishing the self.det_objs through the topic "pub_track_info" which is defined at the __init__ function. Figure 3.17 is the test to check this topic has correct message. And the move base terminal will subscribe this topic and move according to the messages.



```
yuanyueyou@yuanyueyou-911S:~$ rostopic list | grep pub_track_info
/pub_track_info
yuanyueyou@yuanyueyou-911S:~$ rostopic echo -n1 /pub_track_info
tracking_objects:
  -
    track_class: "person"
    track_id: 1
    center_x: 500
    center_y: 162
    distance: 4.08194637299
    color_b: 83.3000717163
    color_g: 86.3500518799
    color_r: 89.8194732666
  -
    track_class: "person"
    track_id: 2
    center_x: 297
    center_y: 174
    distance: 4.4087395668
    color_b: 99.8191604614
    color_g: 101.392959595
    color_r: 103.045257568
---
```

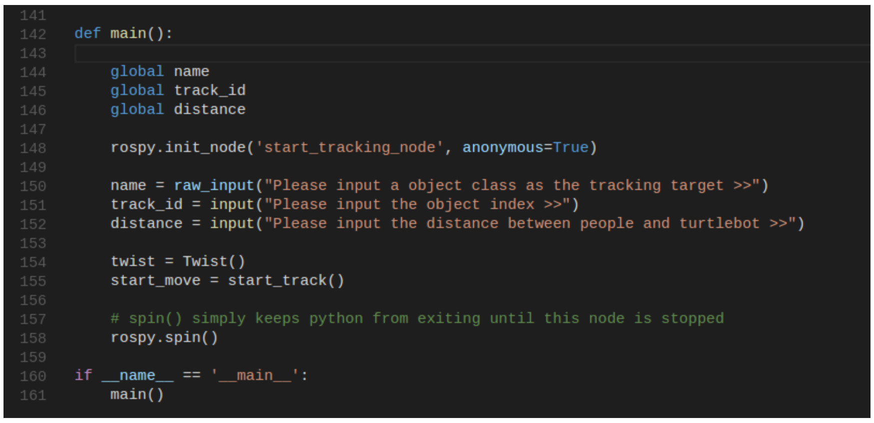
Figure 3. 17 The topic and the message in this topic

3.7 Robot tracking

This section is the last part of this project and all the logic and functions will be described. And here is a introduction about what function is required to be satisfied in this section. And this part is called move_base part, and all the codes related move_base part are written in start_tracking_yolo.py.

After running the detection part code which is real_time_yolo.py, the topic " pub_track_info" will be published with important detected objects information and at the same time there will a screen showing the real time detection results similar to Figure 3.15.

The tracking part needs to subscribe the "pub_track_info" topic which is published by the image_converter class at the detection terminal. And then according to the real time detection result shown on the screen, users need to choose one object as the tracking target. To identify the target, users are acquired to input three values, object class, object index and the desired distance. For example, there are two person showing on Figure 3.15, if the user want to choose person 1 as the tracking target, the object class is the class name of object, here is "person". And the object index is the track_id, here "1" is desired to be input, then the desired distance is how far the robot need to keep from the target in meter. If the input desired distance is input as 3, the robot will adjust its position to keep 3 meters away from person 1 by moving forward or drawback. Besides, if the target disappears suddenly, due to maybe there is a cornet obstacle the vision of camera to detect person 1, the turtlebot will move to the location where the person 1 latest appears. And then the robot will turn until the person 1 is re-detected or the it turns a round. Because, if there is no person 1 appear for a round, it can be seen that the robot miss the target and tracking mission failed. And here are all the functions in move_base section. Figure 3.18 shows the main() function of the move_base part.

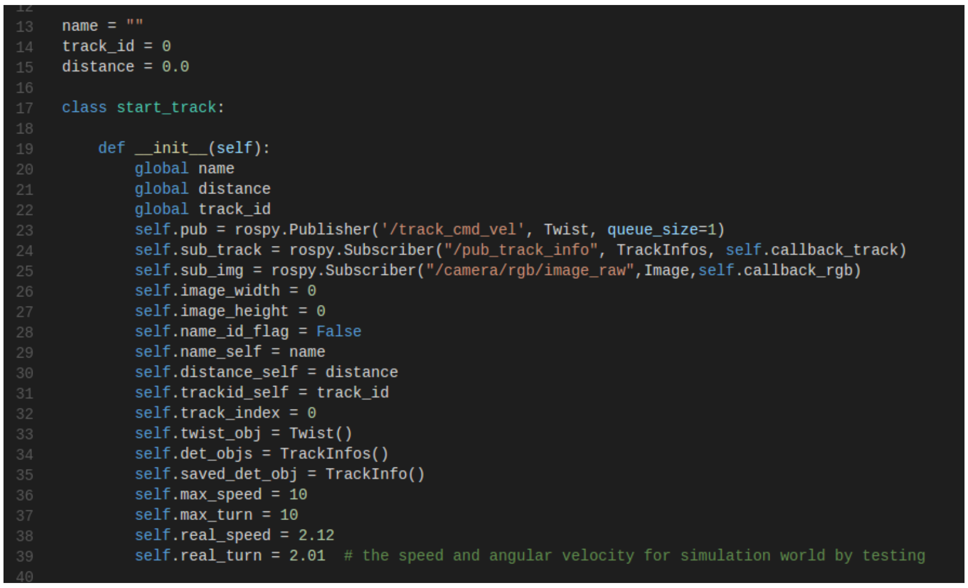


```

def main():

    global name
    global track_id
    global distance

    rospy.init_node('start_tracking_node', anonymous=True)

    name = raw_input("Please input a object class as the tracking target >>")
    track_id = input("Please input the object index >>")
    distance = input("Please input the distance between people and turtlebot >>")

    twist = Twist()
    start_move = start_track()

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    main()
```

Figure 3. 18 The main function of move_base part written in start_tracking_yolo.py

As shown in Figure 3.18, there are two variables defined, twist and start_move. The first one, twist is Twist type object which are used to control the movement of turtlebot and the second one, start_move is a new class defined in this python file which are used to satisfy all the functions at the move base part. And the global variables name, track_id and distance are convenient for being used in start_track() class.

Then turn to class start_track which is the main part of the move_base section. Firstly, for the initial function shown in Figure 3.19, there are two subscribers and one publisher are defined. The sub_track subscriber listens the "/pub_track_info" topic and wait the detected objects information, by using received messages to do the tracking task. And the publisher publishes twist variable which defined in Figure3.16 through the topic '/track_cmd_vel' to control the movement of turtlebot. For the sub_img subscriber listen the topic "/camera/rgb/image_raw" to wait the Image message. This sub_img subscriber is used to assign the values to the member variables image_width and image_height. And according to these two variables are related to keep the target at the center in vision.



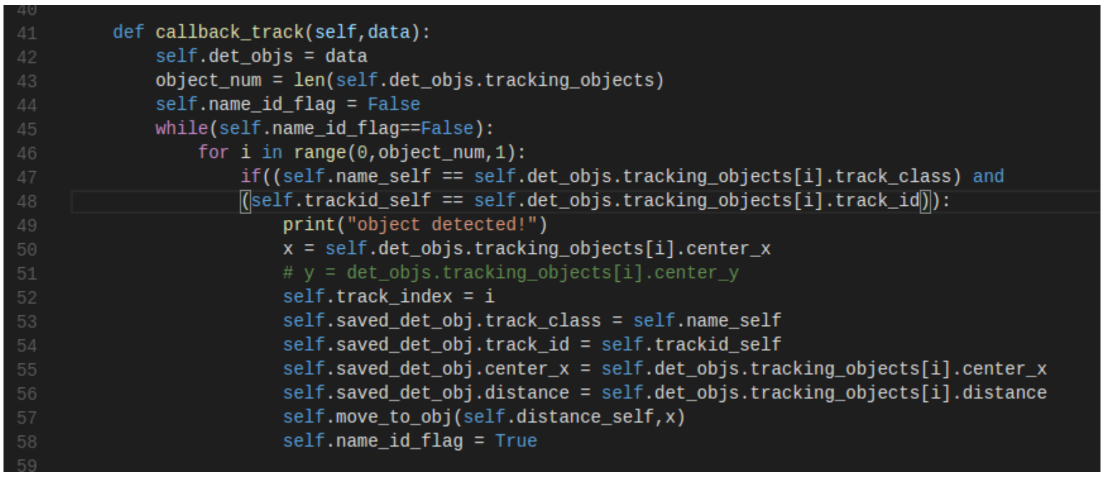
```python
name = ""
track_id = 0
distance = 0.0

class start_track:

    def __init__(self):
        global name
        global distance
        global track_id
        self.pub = rospy.Publisher('/track_cmd_vel', Twist, queue_size=1)
        self.sub_track = rospy.Subscriber("/pub_track_info", TrackInfos, self.callback_track)
        self.sub_img = rospy.Subscriber("/camera/rgb/image_raw",Image,self.callback_rgb)
        self.image_width = 0
        self.image_height = 0
        self.name_id_flag = False
        self.name_self = name
        self.distance_self = distance
        self.trackid_self = track_id
        self.track_index = 0
        self.twist_obj = Twist()
        self.det_objs = TrackInfos()
        self.saved_det_obj = TrackInfo()
        self.max_speed = 10
        self.max_turn = 10
        self.real_speed = 2.12
        self.real_turn = 2.01  # the speed and angular velocity for simulation world by testing
```

Figure 3. 19 The init function of class start_track and its member variables

In addition, the member variable self.saved_det_obj is defined as the TrackInfo type. And this variable is used to save the tracking target information. Once the input values match the received objects, the saved_det_obj will save the target information.

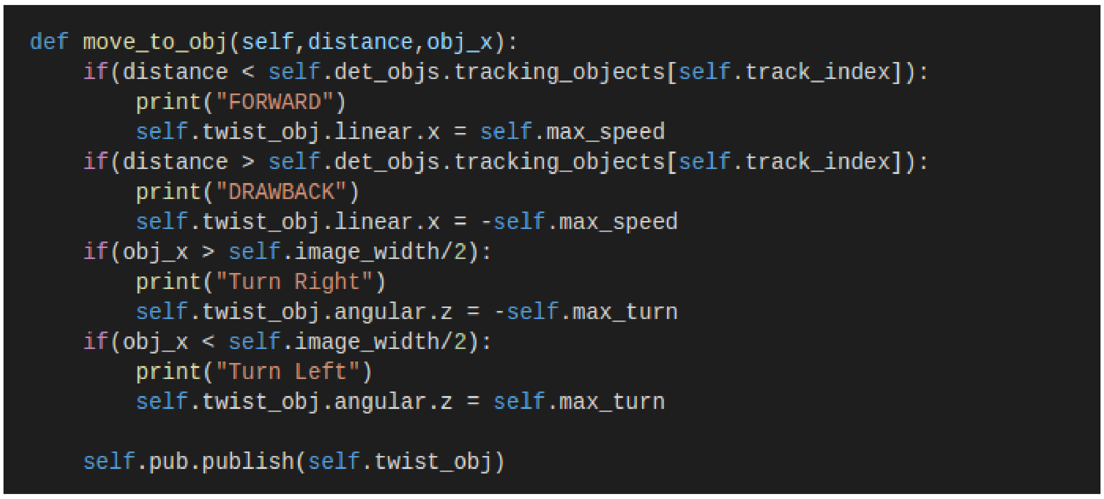
Besides, notice that there are two group of speed and turn. The one group is self.max_speed and self.max_turn, the other is self.real_speed and self.real_turn. The first group is the value assigned to twist object which is used to control the movement. And the second group is the real speed and real turn speed due to the given twist object values in simulation world. These two values are got by moving the turtlebot in simulation world with twist.linear.x = 10 and twist.linear.angular.z = 10 (10 is enough for turtlebot to speed and turn with maximum velocity). Find the move_time and turn_time that turtlebot moves forward 1 meter and turns a round. Then the real_speed = 1 meter/ move_time, real_turn = 2*pi / turn_time.

Then turn to the callback_track function which is the callback function of sub_track subscriber. Part of codes are shown in Figure 3.20. After inputing the name class, track index and the desired distance, then check whether the input values match one of detected objects in the received messages. First find the number of elements from the received data matrix. And transversal all the elements which are actually the detected object information to check whether the input values match one object. If the target object is found from received objects matrix (here save the objects information to self.det_objs), then call the move_to_obj() function to keep the target at center screen and keep the desired distance. The move_to_obj() function is shown in Figure 3.21. And this target information is saved into the member variable self.saved_det_obj. So from here, once the self.saved_det_obj.track_class is not empty and the track_id is not equal 0, there is a detected object matching the input values. Or there was a detected object matching the input values, but now this object information can not be received which means the matched object disappear from vision. The name_id_flag turns to True at the same time.



```
    def callback_track(self,data):
        self.det_objs = data
        object_num = len(self.det_objs.tracking_objects)
        self.name_id_flag = False
        while(self.name_id_flag==False):
            for i in range(0,object_num,1):
                if((self.name_self == self.det_objs.tracking_objects[i].track_class) and
                (self.trackid_self == self.det_objs.tracking_objects[i].track_id)):
                    print("object detected!")
                    x = self.det_objs.tracking_objects[i].center_x
                    # y = det_objs.tracking_objects[i].center_y
                    self.track_index = i
                    self.saved_det_obj.track_class = self.name_self
                    self.saved_det_obj.track_id = self.trackid_self
                    self.saved_det_obj.center_x = self.det_objs.tracking_objects[i].center_x
                    self.saved_det_obj.distance = self.det_objs.tracking_objects[i].distance
                    self.move_to_obj(self.distance_self,x)
                    self.name_id_flag = True
```

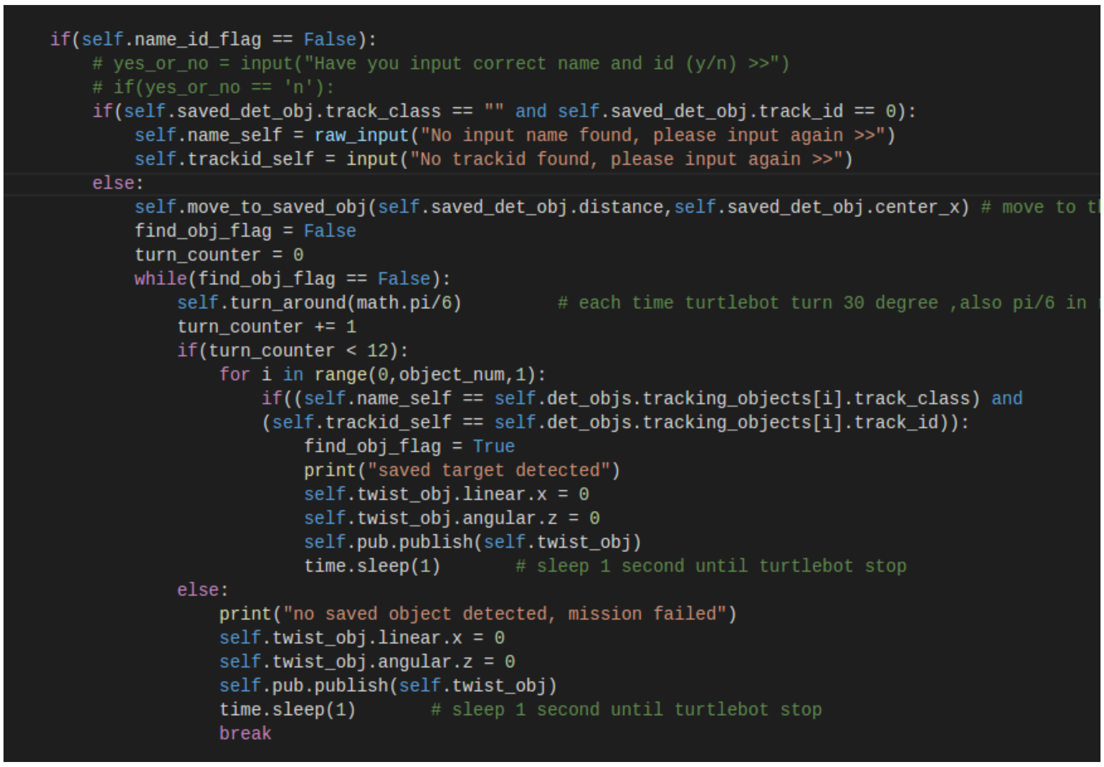
Figure 3. 20 The code show the process about the input target is found in received messages



```
def move_to_obj(self,distance,obj_x):
    if(distance < self.det_objs.tracking_objects[self.track_index]):
        print("FORWARD")
        self.twist_obj.linear.x = self.max_speed
    if(distance > self.det_objs.tracking_objects[self.track_index]):
        print("DRAWBACK")
        self.twist_obj.linear.x = -self.max_speed
    if(obj_x > self.image_width/2):
        print("Turn Right")
        self.twist_obj.angular.z = -self.max_turn
    if(obj_x < self.image_width/2):
        print("Turn Left")
        self.twist_obj.angular.z = self.max_turn

    self.pub.publish(self.twist_obj)
```

Figure 3. 21 The move_to_obj() function

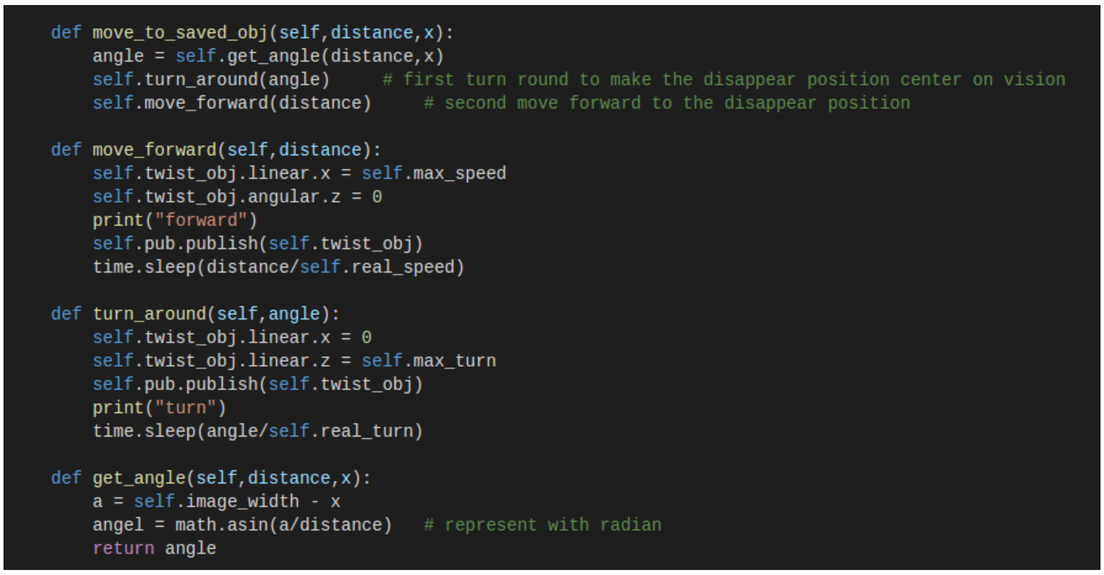
However, if there are no received objects matching the input values from keyboard (equivalent to the name_id_flag is still False after transversal all the objects), there are two occasions occurred. The first one is there is no detected objects matching the input value, the input values may be wrong. And check the self.saved_det_obj.track_class and its track_id, if these two values are empty and 0(equal the default initialization values) respectively, it shows that user inputs wrong values and requires user input name and track index again. And the other occasion is the subscriber can not receive the matched object information. This object may disappear suddenly due to the corner. And the robot will find the latest self.saved_det_obj values, and start to turn around to be opposite directly to the disappear position. Then move forward for self.saved_det_obj.distance. The functions are shown in Figure 3.22.



```
if(self.name_id_flag == False):
    # yes_or_no = input("Have you input correct name and id (y/n) >>")
    # if(yes_or_no == 'n'):
    if(self.saved_det_obj.track_class == "" and self.saved_det_obj.track_id == 0):
        self.name_self = raw_input("No input name found, please input again >>")
        self.trackid_self = input("No trackid found, please input again >>")
    else:
        self.move_to_saved_obj(self.saved_det_obj.distance,self.saved_det_obj.center_x) # move to t
        find_obj_flag = False
        turn_counter = 0
        while(find_obj_flag == False):
            self.turn_around(math.pi/6)         # each time turtlebot turn 30 degree ,also pi/6 in
            turn_counter += 1
            if(turn_counter < 12):
                for i in range(0,object_num,1):
                    if((self.name_self == self.det_objs.tracking_objects[i].track_class) and
                    (self.trackid_self == self.det_objs.tracking_objects[i].track_id)):
                        find_obj_flag = True
                        print("saved target detected")
                        self.twist_obj.linear.x = 0
                        self.twist_obj.angular.z = 0
                        self.pub.publish(self.twist_obj)
                        time.sleep(1)       # sleep 1 second until turtlebot stop
            else:
                print("no saved object detected, mission failed")
                self.twist_obj.linear.x = 0
                self.twist_obj.angular.z = 0
                self.pub.publish(self.twist_obj)
                time.sleep(1)       # sleep 1 second until turtlebot stop
                break
```

Figure 3. 22 The code for no name and tracking index matching objects

Here is a move_to_saved_obj() function, and in this function the member variables self.real_speed and self.real_turn are used. The related functions are shown in Figure 3.23.



```
def move_to_saved_obj(self,distance,x):
    angle = self.get_angle(distance,x)
    self.turn_around(angle)     # first turn round to make the disappear position center on vision
    self.move_forward(distance)     # second move forward to the disappear position

def move_forward(self,distance):
    self.twist_obj.linear.x = self.max_speed
    self.twist_obj.angular.z = 0
    print("forward")
    self.pub.publish(self.twist_obj)
    time.sleep(distance/self.real_speed)

def turn_around(self,angle):
    self.twist_obj.linear.x = 0
    self.twist_obj.linear.z = self.max_turn
    self.pub.publish(self.twist_obj)
    print("turn")
    time.sleep(angle/self.real_turn)

def get_angle(self,distance,x):
    a = self.image_width - x
    angel = math.asin(a/distance)   # represent with radian
    return angle
```

Figure 3. 23 The move_to_saved_obj() function and the related functiona

The other functions are easily to understand, here mention the get_angel() function. The get_angle() function has two input values which are the distance and x. Distance is the distance between the object and camera, here the distance = self.saved_det_obj.distance and the x is the object coordination only for column. The calculation of angel can be get by using the equation below, and the figure 3.24 shows the calculation logic, angle = arcsin((width/2-x)/distance)

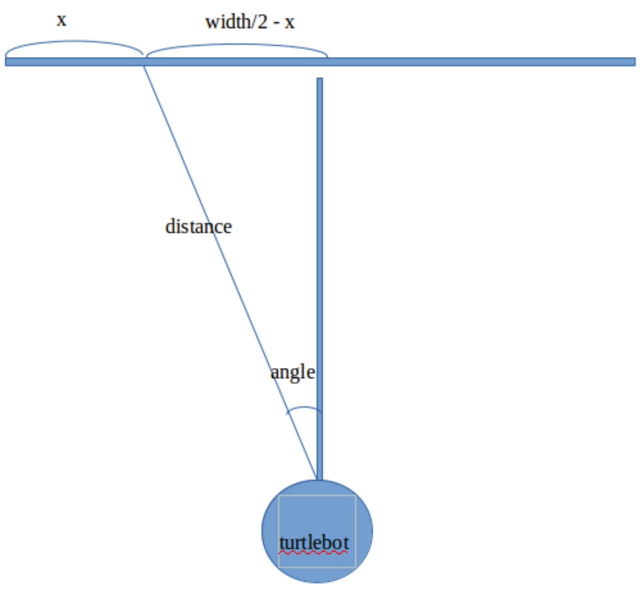
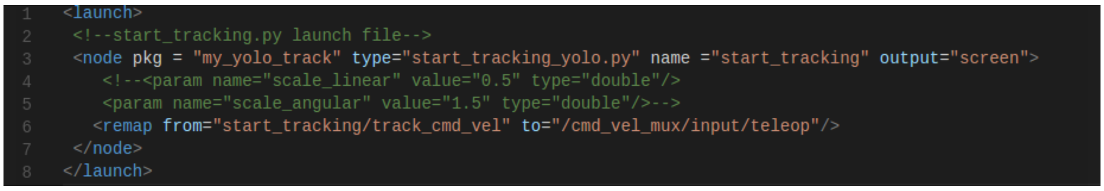


Figure 3. 24 The diagram of how to get angle

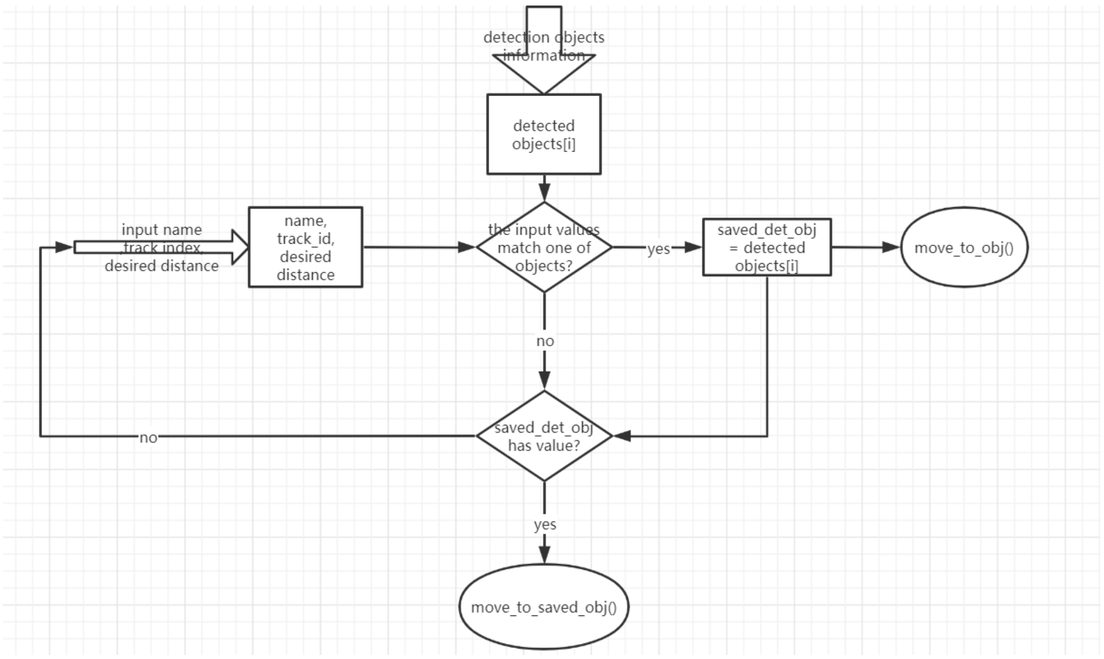
Up to now, all the functions related move_base section are finished, and then this 'start_tracking_node' will publish the '/track_cmd_vel' topic to control the movement of turtlebot. But it is not enough by now, because the turtlebot subscribe the '/cmd_vel_mux/input/teleop' topic to move. So here, it is necessary and convenient remap the '/track_cmd_vel' topic to the desired '/cmd_vel_mux/input/teleop' topic in launch file. Then once the twist object changes in start_track class, it will influence the movement of turtlebot and the move_base section can control turtlebot moving. And figure 3.25 shows the launch file of remap the topic.



```
<launch>
 <!--start_tracking.py launch file-->
 <node pkg = "my_yolo_track" type="start_tracking_yolo.py" name ="start_tracking" output="screen">
    <!--<param name="scale_linear" value="0.5" type="double"/>
    <param name="scale_angular" value="1.5" type="double"/>-->
   <remap from="start_tracking/track_cmd_vel" to="/cmd_vel_mux/input/teleop"/>
 </node>
</launch>
```

Figure 3. 25 The start_tracking_robot.launch file for remapping

Figure 3.26 shows all the logic flow chart of the move_base section.



detection objects information
detected objects[i]
input name track index desired distance
name, track_id, desired distance
the input values match one of objects?
yes
saved_det_obj = detected objects[i]
move_to_obj()
no
saved_det_obj has value?
no
yes
move_to_saved_obj()

Figure 3. 26 Logic flow chart of move_base section

4. Conclusions

This chapter will mainly do the estimation about the performance of these technologies which applied in this project. And find the strength and the weakness for each technology.

First, for the yolov3 detection, this algorithm is very powerful. It can not only detect the default 80 classes, users can also train their own class to add the new class into the coco.name classes list. However, this algorithm require large amount of image processing, and it can caused very low FPS. As shown in Figure 3.13, there is only 0.62 FPS for yolo detection running in CPU. Obviously, this performance is not satisfied for high speed required project. And the low speed of yolo detections by CPU also influence the move_base section. There will be a lag for the detection result comparing with the real time and then the robot will move out of the range and then miss the target. Of course, there are some methods to improve the detection speed. One is to use the GPU to run the yolo detection, its speed can be much higher which can approach to 20 FPS and then the video will show much fluently in real time. Another method is to apply the yolov3-tiny weight file, which can also improve the detection speed, but the yolov3-tiny does not support very accurate object detection.

Second, for the re-id technology, this project applies the detected objects color information as the re-identification character, this method has very high running speed, and it will cost very short time to get the result. But the shortage is that this method is not very accurate, if there are two people with same color clothes, the re-id technology may regard the two people as the same one. Apart from the same color clothes, the similar values of three-primary colors are not accurate. Because, in this method, the color information is actually an average color value got by adding all the color_rgb values and then dividing the total number of pixels. Once the total value of color information is closed with same number pixels, the two different persons may

be regarded as the same one though they wear different color clothes.
So for the next step, the re-id method should include more detailed color processing and do the movement prediction, then the re-id can be more accurate with less limitation.

Third, for the move base section, when there is no target information received due to the object disappear suddenly, this turtlebot will move to the latest position where the target disappeared. However, during the moving period, this turtlebot has its own speed, once the target disappear, the turtlebot can not stop immediately and then cancel all the errors to move forward to the disappearing position. The turtlebot needs time to adjust its movement due to the acceleration, and when the turtlebot stops and starts to go to the disappearing position according to the saved_det_obj distance and coordination, these information is not the newest, there is error occurred due to the continuous movement, speed and acceleration. To solve this problem, the localization technology is need to be applied, the turtlebot needs to calculate its self position and also update the disappearing position continuously. According to the newest information, it is accurate to move to the destination.
Besides, the obstacle avoidance can also be implied in this project for much more useful tracking robot.

5. References

[1] Redmon, Joseph and Ali Farhadi. “YOLOv3: An Incremental Improvement.” *ArXiv* abs/1804.02767 (2018): n. pag.

[2] Wang, S. & Tang, W., 2019. Object Detection in Specific Traffic Scenes using YOLOv2.

[3] Juan Du. Understanding of Object Detection Based on CNN Family and YOLO[C]. Asia Pacific Institute of Science and Engineering. Proceedings of 2nd International Conference on Machine Vision and Information Technology (CMVIT 2018).Asia Pacific Institute of Science and Engineering

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection

[5] Redmon, Joseph and Ali Farhadi. “YOLO9000: Better, Faster, Stronger.” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016): 6517-6525.

[6] "YOLO Real time detection on CPU - Pysource", Pysource, 2020. [Online]. Available: https://pysource.com/2019/07/08/yolo-real-time-detection-on-cpu/.

[7]2020.[Online].Available:https://github.com/opencv/opencv/blob/master/samples/dnn/models.yml

[8] J. Redmon, "YOLO: Real-Time Object Detection", Pjreddie.com, 2020. [Online]. Available: https://pjreddie.com/darknet/yolo/.

[9] "TurtleBot", Turtlebot.com, 2020. [Online]. Available: https://www.turtlebot.com/

[10] "Learn TurtleBot and ROS", Learn.turtlebot.com, 2020. [Online]. Available: http://learn.turtlebot.com/.

[11] "opencv-python", PyPI, 2020. [Online]. Available: https://pypi.org/project/opencv-python/.

[12] "Search for person | 3D Warehouse", 3dwarehouse.sketchup.com, 2020. [Online]. Available: https://3dwarehouse.sketchup.com/search/?q=person&login=true.

[13] "ROS For Beginners LEARNING PATH | The Construct", The Construct, 2020.

[Online]. Available: https://www.robotigniteacademy.com/en/

6. Appendices

Meeting Note

Day of Meeting: 2019/9/13

Semester of learning: 1

Week of Meeting: 1

Timing of Meeting: 16:00 – 17:00

Venue of Meeting: Oriam Hall 2

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

- The supervisor introduced the project topic for each member
- The supervisor described some electric components which may be used
- And assigned the learning task, for the first month, members need to learn how to uses ROS from the ROS wiki website

To do:

- Follow the ROS wiki website and get to know what is ROS, what the ROS can do and some definitions used in ROS.

Meeting Note

Day of Meeting: 2019/9/29

Semester of learning: 1

Week of Meeting: 2

Timing of Meeting: 16:00 – 17:00

Venue of Meeting: PG305

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

- Everyone talked about the learning process of ROS and asked some questions about ROS
- Talk about the project topic that we want to do
- For better to find the project suitable for ourselves, the supervisor showed some existed finished projects and their achievement related to robotics.

To do:

- Keep following the ROS wiki website and learning ROS.
- Think about what project I want to do.
- Read some projects introduction and try to get inspiration from these projects.
- Write some ideas and their introduction. And let supervisor do the difficulty estimation for next meeting.

Meeting Note

Day of Meeting: 2019/10/9

Semester of learning: 1

Week of Meeting: 4

Timing of Meeting: 16:30 – 17:30

Venue of Meeting: PG305

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

- Everyone talked about their ideas about the project they want to do.
- Supervisor gave the estimation of each project ideas and showed what kinds of robot he could provide. Besides he also chose one idea which is interesting and have suitable difficulty for us from all the ideas we provide.
- Supervisor pointed out the difficulties for each project and suggested us to read more paper related our project
- From this meeting, I decided my project which is a tracking robot. This tracking robot can do the people detection and people tracking. The obstacle avoidance function may be included during the tracking period if the time is enough.

To do:

- Keep following the ROS wiki website and learning ROS.
- Find some papers and websites related to the tracking robot. And find what technologies the tracking robot needs.

Meeting Note

Day of Meeting: 2019/10/17

Semester of learning: 1

Week of Meeting: 5

Timing of Meeting: 16:00 – 17:00

Venue of Meeting: PG305

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

- Everyone updated the learning process. And by now I can know the ROS, and understand the ROS communication methods.
- Supervisor and members talked about the issues that there will be a presentation for every student to describe the project they will do.
- Each students will have 5-10 minutes to do introduction of their project, and 5 minutes for answering supervisors' questions. And a Gantt chart is supposed to be included in the PPT to show the timetable for project.

To do:

- Keep following the ROS wiki website and learning ROS.
- Do the PPT about the introduction and do the rehearsal.

Meeting Note

Day of Meeting: 2019/11/1

Semester of learning: 1

Week of Meeting: 7

Timing of Meeting: 10:30 – 11:30

Venue of Meeting: EM2.04

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

- Supervisor gave the feedback of last week presentation.
- Supervisor also showed what kind of robot models can we use in the simulation world. For my project, the turtlebot model can be used.
- Solve a problem about if the detected object disappear suddenly, how the robot can do re-detection and recognize the target as the original tracking target. Answer: By using re-id(re-identification technology), the robot can recognize the target is pervious one if they have very similar character.

To do:

- Get the knowledge about the turtlebot and download in ROS and test the function.
- Find the paper about re-id technology

Meeting Note

Day of Meeting: 2019/11/14

Semester of learning: 1

Week of Meeting: 9

Timing of Meeting: 16:00 – 17:00

Venue of Meeting: EM2.04

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

Solve some problems

- When testing the gmaping function by using turtlebot, the turtlebot can not gmap perfectly.

  Solving: Simulation in gazebo is not accurate. In real world, gmaping will be better. Can also add hokuyo in turtlebot for better simulation.
- There are some packages which are supported in indigo. Some packages are not supported in kinetic, how to deal with this problem

  Solving: there definitely are some different packages between indigo and kinetic, because they are the totally different ROS version. However, there are still most of packages are available in kinetic. And kinetic is enough to support the functions of the tracking robot.

To do:

- Keep on testing the functions of turtlebot.
- Try to add hokuyo sensor on turtlebot in simulation world.

Meeting Note

Day of Meeting: 2019/11/22

Semester of learning: 1

Week of Meeting: 10

Timing of Meeting: 10:30 – 11:30

Venue of Meeting: EM2.04

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

Solve some problems

- Turtlebot support indigo system, part of functions can not satisfied in kinetic system.such like some packages ros-indigo-turtlebot-concert, ros -indigo-gazebo-concert, ros -indigo -rocon-remocon, ros -indigo -concert-services → also support kinetic system, just need to find the available packages
- no package of ros-indigo-hokuyo-node
  Solving: there is Urg node. What is the difference between hokuyo and urg→ the new version of hokuyo node, and available for kinetic system
- change the fake laser scan: modify the remap section of the launch file to remap the fake scans to another topic, what is remap → change the topic name to another topic name
- Rviz global status error→ need change the link to /base_link
- Tf and urdf, brief introduction → need to be known, used to config the models describe the links and joints and, tf is consist of joints and links with

coordinates

To do:

- Try to finish the function of robot tracking
- following the ROS ignite website to learn more about ROS, python, TF and URDF.

Meeting Note

Day of Meeting: 2020/2/26

Semester of learning: 2

Week of Meeting: 19

Timing of Meeting: 16:00 – 17:00

Venue of Meeting: EM2.04

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

Solve some problems

- In the launch file and in the node tag, the following of type tag should be the excuteable file such like the move_robot.py ,or .cpp file but the following type do not have the character of a excuteabe file, it is like a folder. Why?

  <node pkg="aloam_velodyne" type="ascanRegistration" name="ascanRegistration" output="screen" />

  → this should be the cpp file without .cpp , and there are some occssians that python file without .py

- How to download the packages on the real turtlebot and how to config?

  →download all the function packages from computer to turtlebot and install the driver on computer to connect computer with turtlebot.

- How to change the color of walking person the gazebo model included in gazebo library.

  →Need to change the .dae file in the mesh folder. Or change the .sdf file but the .sdf file is similar to the urdf file with the description of links and joints, and also some visual tag. In the visual tag may include the local file //model/mesh/walking_person.dae. so in this situation, also need to change the dae file.

To do:

- Try to finish the function of robot tracking
- Test the solving methods which provided in meeting.

Meeting Note

Day of Meeting: 2020/3/10

Semester of learning: 2

Week of Meeting: 21

Timing of Meeting: 12:00 – 13:00

Venue of Meeting: video meeting

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

Solve some problems

- Is it matter that define different node name from .py file and .launch file? For example, in the .py file, a node named "move_robot_node" is initial by using rospy.init("move_robot_node"),
  Then in the launch file, in the node tag, the name of node is "start_tracking_robot". Finally the start_tracking_robot node can be shown in rqt_graph, but there is no node called move_robot_node defined in .py file before
  Solving: In python file, the initial node name is a default node name, and we can rename this node for another name in the launch file just using the method shown above. And if we do not want change the name, and just input the command : rosrun my_tracking_robot start_tracking.py . in the rqt_graph, there will be a move_robot_node node shown here
- A much more convenient detection method is introduced, the yolo detection.
- Use the rgb image to do the detection and use depth image to get distance.

To do:

- Apply the YOLO detection in project.
- Try to use yolo to detected the received messages from topic "camera/rgb/image_raw"
- Try to get the distance from the "camera/depth/image_raw" topic

Meeting Note

Day of Meeting: 2020/4/17

Semester of learning: 2

Week of Meeting: 26

Timing of Meeting: 10:30-11:30

Venue of Meeting: video meeting

Present: Sen Wang Signature: Sen Wang

Yuanyue You

Kaicheng Zhang

Zhuocheng Han

Zhanheng Li

Ziyu Du

Agenda

Solve some problems

- If there are two or more topics needed to be subscribed, how can we achieve this function in the same node?

  Just define two subscribers, and the two subscribers will listen these two topics rather than creating a new thread.

- How to change the dae file to change the color of people

  Use the collada to change the dae file, or just download a new people model from website.

- How to get the distance information from /depth image raw topic.

  Transfer the value received from depth topic to cv_image by using cv_image = bridge.imgmsg_to_cv2(data, “32FC1”). Here cv_image contains the distance information in meter for each pixel.

To do:

- Finish all the functions of tracking robot.