**Objective:**

An autonomous balloon tracking vehicle capable of tracking a balloon and terminating it.

For ECE 5725 Design with Embedded Operating System final project, we use a Raspberry Pi to control a robotic robot to autonomously track a falling red balloon and follow it until popping the balloon. The camera can move up and down by tilt kit and the wheel could move around in the ground. Our robot can also be remotely controlled by pygame interface in computer connecting with it in different modes.

**Introduction:**

We create a tracking robot with Raspberry Pi that can track a ~~moving and~~ falling red balloon and try to pop the balloon before it lands. We implemented the color detection to recognize the balloon by OpenCV. PID algorithm could calculate the suitable parameters to control the tilt kit of camera by hardware PWM precisely and stably and the movement of wheels by DC controller. Besides, python multiprocessor modules are implemented to decrease the latency which allows us to simultaneously capture video, process the images and calculate the parameters of PID control and send the command to the motors.

Based on Raspberry Pi, we create a tracking robot that can track a falling red balloon and try to pop the balloon before it lands. In order to recognize the balloon, we implemented the color detection by OpenCV. The precise and stable locomotive of the robot, which includes camera tilt kit and two DC motors, has been controlled via PID algorithm. Besides, leveraging python’s multiprocessor modules tremendously decreases image processing latency in real-time.

**Design and Testing:**

We only use one pi camera to detect objects and measure the distance between the object and the robot. Our solution is to move the robot according to the location of the balloon in the images and the criterion is to keep the balloon in the center of the images, which means the direction of moving robot is in the line with that of the balloon. Additionally, the area of the balloon in the image can reflect the value of the distance between them. So it requires the accuracy of the recognition to the balloon.

Since we only have one Pi camera, we couldn’t measure the distance from the method of two camera. Our solution is to keep the balloon on the center of the image as possible. Therefore, the area would represent as the distance between the balloon and the robot. The Y axis differential would be the rotation error. And the X axis differential would be the tilt error. Consequently, our robot should require the accurate recognition on the balloon.

In order to accomplish the project better, the best approach was to divide it into various smaller parts individually and combine them at the end. The main task is hardware and software part, which are tackled different tasks as following introduction.

To accomplish the project, we have divided it into several smaller tasks and combine them in the end. In general, there are two parts which are hardware and software. And they have different tasks to be tackled in the following introduction.

**Hardware:**

Our robot, in short, is a differential drive vehicle with a Pi camera on it. Therefore, we would need two motors to drive the robot and a ball bearing keep it balancing. We also need robot frames to hold everything up. Here’s our shot of the robot.

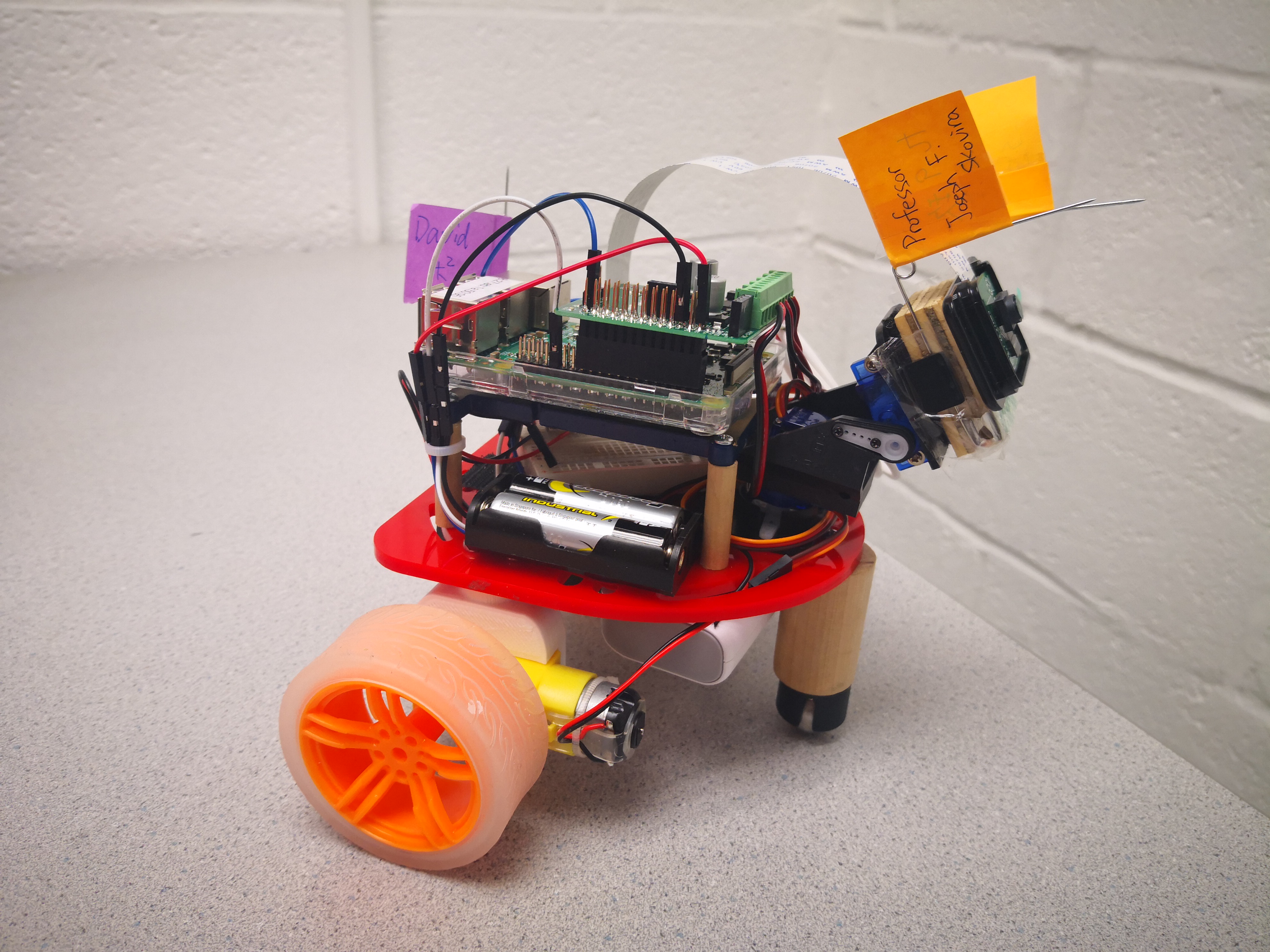
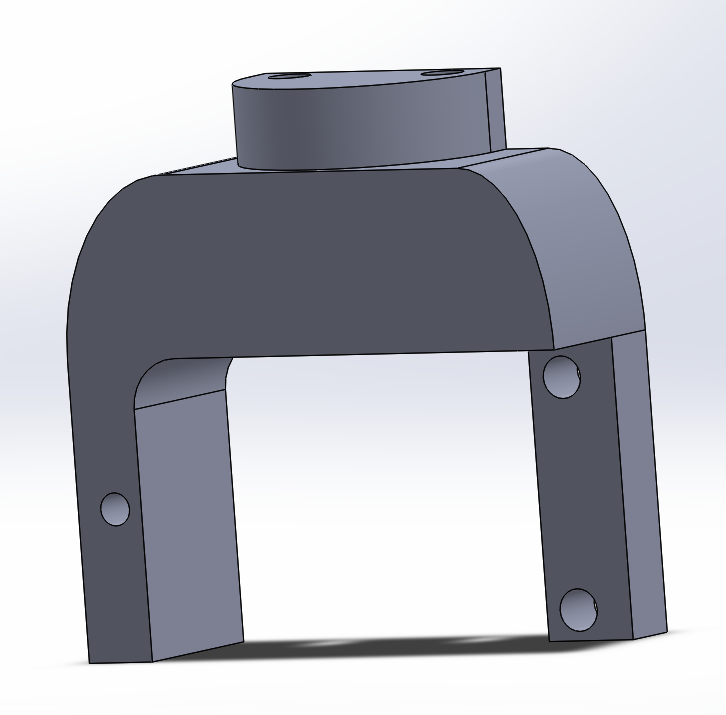
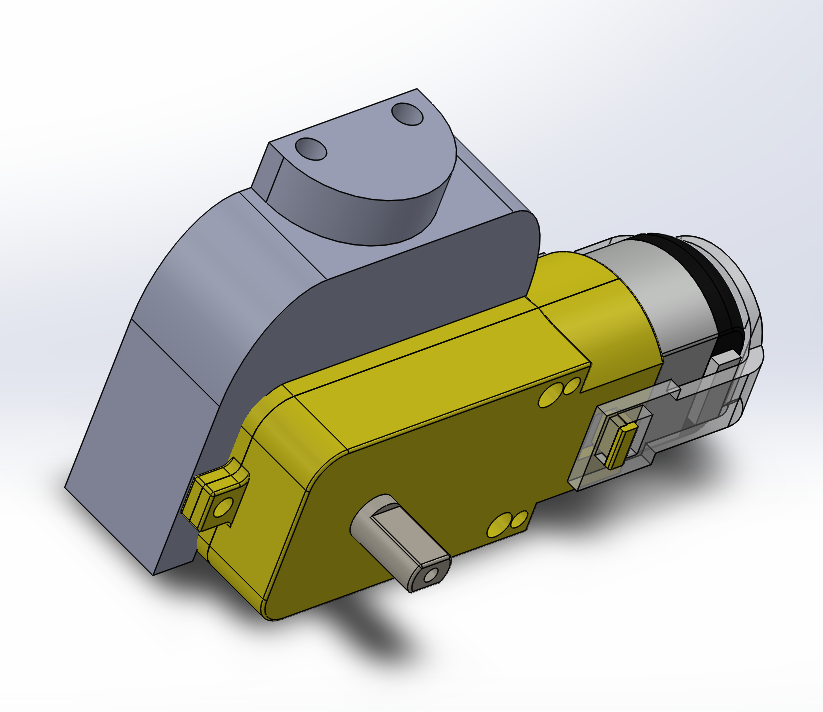


Figure X. robot appearance

1. Self-designed motor bracket

Everything is in placed from the previous lab section, except the motor brackets. We self-designed the motor brackets using Solidworks. After several simulation and testing, we would be able to fit the motor with the robot frame properly.



1. DC motor driver

Due to the speed limit, the Parallax continuous rotation servo is replaced by TT DC Gearbox Motor with higher rpm (200RPM). However, unlike Parallax continuous rotation servo, we can’t control the DC motor with PWM commands. The DC motor is powered from 3VDC to 6VDC. The higher voltage, the faster it goes. As a result, we bought a motor driver – “RasPi Robot Board v3”. It could transform the PWM signal into voltage level and thereby control our DC motors.

The driver is best required with 9VDC input to supply the motors, although a motor only required 5DV input. This is because if the voltage input is lower than 9VDC, the driver could be malfunctioning. Any unexpected result would come out of nowhere and it’s a painful bug to be realized when developing the robot.

Also, keep in mind that although 6VDC is its maximum speed limit, it is when the DC motor is unloaded. Moreover, the higher voltage it required, the more current it would need as well. In terms of that means it’s more power consumption than normal, and we need to keep changing the batteries for stable speed outcome and high rpm.



**Software:**

1. **Object Recognition Algorithm:**

Our object recognition algorithm could detect the red objects videoed by the Pi camera and calculate the center position. The usage of the OpenCV library makes the algorithm easy to implement. First of all, we install the openCV library on our Raspberry Pi. There are some ways to install the openCV library. The easiest and direct method is to run the command “*sudo apt-get install libopencv-dev python-opencv*” in the command prompt, which can help us to save several hours to compile the openCV. The version of our installed openCV is **“2.4.9.1”*。*** After installing the openCV successfully, we need to assemble the Pi camera onto the R-Pi and set it up in configuration of R-Pi. As for the design of the recognition algorithm, it can be broken up into the following parts.

1. Create a video capture object of openCV which can take picture at real time by running “cv2.VideoCapture(0)” and when using “cap.read()”, it will return 640\*480\*3 picture array, which is the default resolution and can fit our requirement of recognition. Three channels is (R, G, B) color space.
2. Run “cv2.cvtColor(frame, cv2.COLOR\_RGB2HSV)” can convert the image from RGB color space to HSV (hue, Saturation, Value) color space. We can benefit from this conversion because we need to threshold the HSV image to get the binary image which only keep the red balloon part by setting the low\_red and upper\_red threshold and it is convenient and easy for us to adjust the threshold in HSV color space.
3. We set the lower\_red is (156, 100, 40) and the upper\_red is (180, 255, 255). Use the syntax of “maks = cv2.inRange(hsv, low\_red, upper\_red)” to do threshold. It will make the pixel values that are less than lower\_red and more than upper\_red equal to zero and the pixel values that are in between them equal to 255.
4. We also use the techniques of computer vision to process the binary images including do “cv2.morphologyEx” of open and close to filter the background noise and make the boundary of binary image smooth and clear.
5. Run “cv2.findContours(mask,cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_NONE)” to do the most important thing which is extract the boundary of the object. If it detects the boundary, it will return a list that contains several calculated contours although most of them are zero. If it does not, it will return a empty list.
6. We need to pick up the longest contour from returned contour list. The method we used is to calculate the area of every contour and find the maximum value that is what we want. Then, calculate the spatial moment that can compute the center position of the object and the area of the object. s
7. **PID control Algorithm:**

The size of image is 640\*480 and thus the center of the image should be (320, 240). The object recognition algorithm should compute the center position of the object. Compute their difference and use it as feedback to control the servo and make the camera stare at the object, thereby achieving the camera tracking. Besides, the motors of wheel also need to be controlled to move suitably to track the balloon.

Our objective is to make the balloon as center as possible. From the respective of the image input, the size of the image is 640\*480 and thus the center of the image should be (320, 240). The image processing would recognize the red balloon, find out the contour, and calculate the center point and area.

Given the balloon’s coordinates and the area, which is the system output, we would be able to control the robot with a closed loop controller. To make a simple closed loop controller, we choose PID controller. The benefits are one that it is simple, and second it is robust enough for our robot.

PID algorithm is a closed loop control algorithm. PID stands for proportional-integral-derivative. It takes the present error, past error, and the future error as inputs to make appropriate control and thus reduce the error.

**Markdown for PID control algorithm**

<li> P-term: proportional to the error. </li>

<li> I-term : proportional to the integral of the error, </li>

<li> D-term: proportional to the derivative of the error. </li>

Actually, not all PID controller used up three terms. To be more specific, the there are controllers like

<li> PD controller, which Ti is unbounded. </li>

<li> PI controller, which Td is zero. </li>

<li> P controller, which Ti is unbounded and Td is zero. </li>

Ti is the period of the integration time, and Td is the period of the oscillation time.

For the reason that I-term is generally considered as dangerous when implementing PID controller, we only choose P controller and PD controller. From X differential

1. **Multiprocessing Algorithm:**

To achieve our goal of tracking a falliing and moving balloon and pop it, we must decrease the latency and running time of every part as far as possible. At the beginning, the recognition and the course of sending order to control the motors and servo took about 105ms, which means it only processed less than ten pictures every second and it is far away from what we should do since we need to track the moving object and we only have limited time to decide the robot’s movement. This delay time is too long. To reduce the delay time, one solution is to parallel process different task because of four cores in R-Pi, we can assign four jobs into four cores. In python, we could use the multiprocessing library to fully take advantage of four cores of R-Pi.

1. The first core is used to focus on capturing the video and put the image it reads into the “send\_frame\_queue”, whose minimum running time is 33 ms because the frequency of the Pi camera taking video is 30 fps.
2. The second core is used to get the frame of images from send\_frame\_queue and process the images and extract the contours. Then put the contours into receive\_contour\_queue, which takes about 10 ms at most.
3. The third core could get the contour from receive\_contour\_queue and compute the center position and area of the object and implement the PID control algorithm and control the motors of wheels. Since we set the sleep time to 20 ms when controlling the DC motors. In addition, it should put the parameters of controlling servo of camera into send\_motor\_queue. And the maximum running time of this processor is 22 ms.
4. The forth core can get the parameters to control the tilt kit from send\_motor\_queue and realize prediction function to ensure the smooth movement of tilt kit , whose maximum running time is 25 ms.

This describes the tasks that every core executes and the time they spend. The longest time is up to the frequency of the Pi camera, which is unchangeable to some degree. So this multiprocessing system has optimized the entire program and decrease the running time every loop from 105 ms to 34 ms. Note that the “print” syntax will cause the delay and if wanting show the image onto the monitors, it will increase delay time.

**这里可以放multi前后的对比视频**

1. **Prediction Algorithm:**

When balloon quickly moved and get rid of the range that the camera can video, the servo of camera will not know how to move and it will be stuck. To solve this problem, we create the prediction algorithm. Prediction Algorithm is to make the servo move at the speed that is slower than the original speed but the same direction. “Original speed” is the speed of robot before the balloon disappears. This is reasonable because the balloon is falling and we assume there is no wind to affect the balloon’s motion.

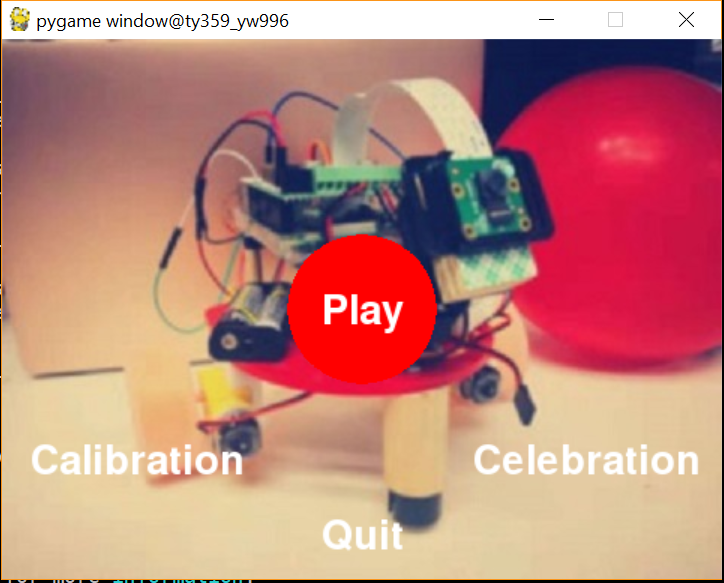
1. **Pygame interface**

There are three modes in tracking robotic robot. We use pygame library to implement visual interface to control them as the following picture shown.

The first mode is calibration mode, which is to find the balloon at the beginning in case the person who toss the balloon standing behind the robot and it will not see the balloon.

The second mode is play mode, which is main program to realize our function. When a person toss a red balloon, the robot will track the balloon until it explodes the balloon.

The third mode is celebration mode, which makes the robot turn around and move forward and backward like a puppy.



**此处需要pygame的界面截图**

**Issue:**

1. Since our robotic robot need to track the falling balloon and move to the landing location very fast and considering that python is a kind of interpreted programming language and C++ is a complied programming language, we initially planned to install C++ version of openCV. We tried to install the C++ OpenCV library several times and every time it took more than three hours to compile it. We finally installed the C++ openCV successfully but when we tried to run some codes, it failed and said that it cannot open the Pi camera, which took us about one week. After discussion with Professor Skovira, we decide to use python openCV first and see whether it can meet our requirement. But we will still introduce how to install C++ openCV here briefly.

First of all, we need to install several required packages that we have not installed before such as CMake 2,8,7 or higher, Git, GCC and GTK+2.x or higher, including headers (libgtk2.0-dev). These packages can be installed in a terminal directly.

Secondly, OpenCV Source Code is needed which can download from the Git Repository by the command “cd ~/<my\_working\_directory>

git clone https://github.com/opencv/opencv.git

git clone https://github.com/opencv/opencv\_contrib.git”

Thirdly, Building OpenCV from Source using CMake.

1. Create a temporary directory, where we want to put the generated Makefiles, project files as well as output file and enter there.
2. Configure it by running “cmake -D CMAKE\_BUILD\_TYPE=Release -D CMAKE\_INSTALL\_PREFIX=/usr/local ..”, which will take more than ten minutes.
3. After finishing this, we need to describe some parameters or during the process of compiling, there will be some errors.
4. From build directory execute “make -j7 # runs 7 jobs in parallel”.
5. Install the library by execute “sudo make install” form build directory.

It should work, but there will definitely be a lot of errors. Good Luck!

1. After python OpenCV library installed, we tried to run testing code, there is one error saying “OpenCV Error: Assertion failed (scn == 3 || scn == 4) in cvtColor, file /build/opencv-U1UwfN/opencv-2.4.9.1+dfsq1/modules/imgproc/src/color.cpp, line 3737”. We searched it on Google, this error can be fixed by running “sudo modprobe bcm2835-v4l2“, which is used to intall the v412 driver on the bcm2835. We thought it is one time install but when the R-Pi reboot, the same error appeared again. Therefore, we add this command into /etc/modules. Then, it can work smoothly.
2. To control the servo of pan tilt kit, under the recommendation of Professor Skovira, we use the hardware PWM to generate control signal instead of software PWM, this is because the signal generated by hardware PWM is stable and less noise in a R-Pi multiprocessing environment. When we following the instruction of the professor to build the hardware PWM file and connect the circuit, we cannot send any signal to the GPIO pin. After asking other students who have experience to implement the hardware PWM, we know some very significant points. One is only GPIO pin 12 and pin 13 can be used to generated hardware PWM signal. Another is before running the code, we need to run “sudo pigpiod” to launch the pigpio library as a daemon.
3. Because of tracking and chasing the falling balloon, we need to faster motors to drive the robotic robot and thus, with the help of the professor Skovira, we decide to use the DC motor to drive and buy DC controller to control two DC motor to move. It works well but we found that once we add the controller program into multiprocessing, the whole program will be stuck and the queue will keep a lot of unprocessed data. After we monitored the running time of every core, we noticed that the every running time of the third core which is used to process the PID control algorithm and send order to DC controller is far longer than others and the average running time is 202 ms. At first, we thought this could be the problem of the hardware because of the delay of opening motors or the bad connection of circuit to make the transmitting slow. But when we check everything about the circuit, there is no bad connection. So we searched the software stuff. We found there is a syntax about “time.sleep(0.2)” in the library that is used to control the DC controller and this is probably the reason why it runs so slowly. So we changed the source of the library and compile it again. It worked, the running time of the third core decreased to 22 ms.

**Reference :**

[1] install C++ openCV

<https://docs.opencv.org/master/d7/d9f/tutorial_linux_install.html>

[2] RpiRobotBoard

<https://www.adafruit.com/product/1940>

[3]