Calibrations on the Ground Robot

Calibrating the Ground Robots Servos

To calibrate the servos on the robot we placed two blue pieces of tape on a piece of paper to serve as positions markers, and placed the paper 5 feet away from the robot. We then positioned the laser so it was in line with the side of one of the markers and took a picture. We then moved the servo to the left 1 degree and snapped another picture. Using Photoshop, we pasted the first image captured onto the background and created a new layer which we pasted the second image captured. We then moved the second layer into position so that the markers and corners of the paper were in line with one another and the two laser positions were side by side as shown in the image below.

Above image is cropped from two actual images

In the image above, there is a black pixel in the center of each laser position which was used to calculate the distance between the two laser points, in pixels. The black pixel located in the laser point on the right side, had a center of point of (272, 216) and the left had a point of (199, 216). This shows that after moving the servo 1 degree, the position of the laser moved 73 pixels.

In the center of the blue tape markers there is a bright blue pixel, used in a similar fashion as the black pixel mentioned above. The marker on the left had a center point of (57, 194) and the marker on the right at (299, 179). The distance between the centers of these markers was found to be 242 pixels. We then found the displacement of the points to by 299/242, and got approximately 1.2. Using this as our baseline, we can approximate how many degrees to move the servo with the following formula and rounding the answer:

Let the pixel distance between two points = X. Given the average error of 1.2 pixels and the average change in pixels of 73, we can then estimate how many degrees of servo movement (S) are needed as:

Horizontal Servos: $S= (X*1.2)/73$

We repeated this process for the vertical servos, and found its error to be 1.8, giving use the following formula:

Vertical Servos: $S=(Y*1.8)/73$

Calibrating the Robots Motor Movements

Each motor on the ground robot is equipped with a speed encoder. The speed encoders are used to measure the number of wheel rotations that occur during the robots movement. This makes it possible for the robot to move straight, turn, and change speeds, by making adjustments to the wheel speed of each motor. This change in speed or movement is achieved by passing in an integer value, called the modifier, to the robots motor run method. The Raspberry Pi handles this by passing the modifier value along to the Arduino when sending a move command. The Arduino will then execute this movement command for the distance specified by the modifier value.

The question then becomes, what modifier value should the Pi send to the Arduino to move the robot one foot in distance? In order to estimate the relation between modifier values used and actual distances traveled, the following calibration was conducted and the results are shown in the tables below.

The ground robot was placed at a starting position which was physically measured to be 10 feet away from the opposing wall. Next, the robot's laser was used to measure the distance from the wall, which was found to be 10.16 feet from the wall. Because the error found between these distances was 0.16 feet and we do not require absolute precision here, this can be ignored. Next, the robot was sent a movement command to move straight with the modifier value of 100 *(Forward Movement Command w/ Modifier Value = 100: mf0100).* After the robot has completed its movements, the distance between the robot and the wall was again measured and recorded in the table below, once manually and once by the robot. This process was repeated for three runs for additional data. The average distance traveled was found from the actual distance traveled of all three runs, which was found to be approximately 1.92 feet, or 23 inches. Next we find the average number of inches travelled per modifier value used, which in this case was found by dividing the used modifier value of 100 by the average number of inches traveled. This resulted in 4.41 inches traveled per modifier value. With our goal to move one foot, we can now adjust the modifier value sent to be 53 (4.41*12). Using the modifier value of 53, we then used it within the Arduino code as the standard value to move the robot 1 foot. The new forward movement command used to move the robot 1 foot is now *mf0001*. The above method can be summarized below.

With this, the Raspberry Pi can now send a specific distance for the Arduino to move the robot. For example, if the 'mf0001' command is sent to the robot, it will move forward one foot. If 'mf0010' is sent, the robot will move forward ten feet.

****Note:** This calibration is no longer accurate, and the robot needs recalibration. In order to address the issue of the robot not driving in a completely straight line, we had to lower the robots ground speed. Therefore, the modifier value of 53 no longer is accurate for 1 ft. of movement on the robot. The easiest way to recalibrate it from this point would be to increase the modifier value in the Arduino code until the robot moves 1 ft. Note that the robots turning was not affected by this.

Test Run	Actual Starting Distance	Found Starting Distance	Distance After Movement	Expected Distance Moved	Actual Distance Moved	Error In Distance Moved
Run One	10 ft	10.16 ft	8.22 ft	1 ft	1.94 ft	+0.84 ft
Run Two	10 ft	10.29 ft	8.36 ft	1 ft	1.93 ft	+0.83 ft
Run Three	10 ft	10.16 ft	8.26 ft	1 ft	1.9 ft	+0.8 ft
Data Recorded Before Calibration *Movement Command Used: 'mf0100'*			**Average Distance Moved:**		1.92 ft	**Avg. Error:** +0.82 ft

Test Run	Actual Starting Distance	Found Starting Distance	Distance After Movement	Expected Distance Moved	Actual Distance Moved	Error In Distance Moved
Run One	10 ft	10.23 ft	9.26 ft	1 ft	0.97 ft	-0.03 ft
Run Two	10 ft	10.16 ft	9.13 ft	1 ft	1.03 ft	+0.03 ft
Run Three	10 ft	10.18 ft	9.22 ft	1 ft	0.96 ft	-0.04 ft
Data Recorded After Calibration *Movement Command Used: 'mf0001'*			**Average Distance Moved:**		0.98 ft	**Avg. Error:** -0.01 ft

Additional Notes on Motor Calibration:

If the speed of the robots motors are altered within the Arduino code, if the wheel encoders are recalibrated, or any changes with the robots distance algorithm are made, the ground robots movements should be recalibrated to maintain consistency. Additionally, in order for the calibration to work properly, the robot must be able to move completely straight in either direction. This requires that the robots speed encoders are calibrated properly. It is also EXTREMELY important to not touch the ground robots laser at any time and any moving of the robot must be done with care because the laser can become uncalibrated and make each distance read taken from the robot invalid.

****Note:** This calibration is no longer accurate, and the robot needs recalibration. In order to address the issue of the robot not driving in a completely straight line, we had to lower the robots ground speed. Therefore, the modifier value of 53 no longer is accurate for 1 ft. of movement on the robot. The easiest way to recalibrate it from this point would be to increase the modifier value in the Arduino code until the robot moves 1 ft. Note that the robots turning was not affected by this change.

Calibrating the Ground Robots Turn Movements

Similarly to how we calibrated the robots movements, the robot makes turns based on the modifier value passed in from the Raspberry Pi which sent the initial command. Our initial goal for this calibration is to get the robot to spin 360 degrees. We simply just tested various values until we found one that met this goal. For the ground robot to turn 360 degrees, we found that sending the modifier value of 88 to be a fair approximation.

Given the above values:

360/88 = ~4.1 or the average modifier value needed to turn per degree desired

So, if we let the Degree Desired = D, and let the Modifier Value = MV we can use the following formula to estimate MV for all turns.

D/ 4.1 = MV

We used this approximation to come up with the table below, which shows the estimated modifier value needed to turn the robot various degrees, which includes the actual degrees turned if our estimate was shown to be incorrect.

Desired Degree of Turn	**Modifier Value Used**	**Actual Number of Commands Needed for a 360 Turn**	**Expected Number of Commands Needed for a 360 Turn**	**Actual Degrees Turned**
360	88	1	1	~360
180	44	2	2	~180
90	22	4	4	~90
45	11	8	8	~45
15	*3*	*19*	*24*	*~19*

As the above table above suggest, the smaller the degree of turn the more inaccurate this estimation becomes simply because we are using approximate values. After testing the values 22 for 90 degrees and 11 for 45 degrees, we found that the robot over-turned slightly. Because of this, we modified the values used in the Arduino code to be 21 for 90 degrees and 10 for 45 degrees. We've also included the following special case for the Arduino when the degree sent is less than 45.

For all other cases, by using the values found in the table above, we derive the following formula to be used on the Arduino.

*(D/45)*11 where D is the degree desired to turn*

Calibrating the Ground Robots Laser

In order to calibrate the robot's laser, we used a piece of standard computer paper with a known aspect ratio and size, mounted on a red sheet of plastic as our marker *(Figure 1).* The marker was setup on one side of the room with the robot placed directly in front of it. The laser on the robot was then mounted at an upward angle on the robot, such that from 5 feet away from the marker, the laser pointed to the bottom of the marker and capture images both with and without the laser turned on. We then moved the robot back further, and repeated these processes until we had various images from 5-25 feet away from the marker. These images served as our test sample, for verifying that the image processing algorithms were working correctly.

Figure 1

We encountered a compatibility issue with the Python module **pySerial** and the **OpenCV** library cv2. The **OpenCV** library was installed on the virtual environment, cv3, which by default does not have access to all of the libraries/modules that are installed on the system wide level. When we installed Python's **pySerial** module, we used the *apt-get* tool, which automatically installs Python modules in the default directory, which is only accessible within the system wide level (from the Pi). When working on the virtual environment cv3, the compiler could not access the **pySerial** module since it was unable to access the systems libraries. To solve this problem, **pySerial** has to be installed on the virtual environment using the *pip-install* tool, which is able to recognize the virtual environment and install it in a way that the module is accessible. An example on how to do this is shown below:

```
$     sudo pip install pyserial
```

The *'DroneUtils.cpp'* and the *'DroneUtils.h'* files, contain all of the image processing algorithms that will be used on the drone and ground robot. In order to use these algorithms in Python, we had to install the **Boost.Python** library on the Raspberry Pi. This allows for Python scripts to call the c++ methods, acting as a wrapper class. Installing **Boost.Python** on the Pi requires that **OpenCV** be installed first. The following command will then install the boost library:

```
$     sudo apt-get install libboost-all-dev
```

With boost installed, the c++ file and header need to be compiled on the Pi. To compile c++ files on the Raspberry Pi, you use the following command:

```
$     make
```

The compiled *DroneUtils* file can now be imported into Python code, allowing the exposed c++ methods to be called in Python. To test out the image processing algorithms for estimating a piece of papers distance based on contour lines and the aspect ratio of the papers size, the positions of an object's corners, and the position of the laser pointer, we used the *'laser10b.png'* and the *'laser10bwithout.png'* images which were captured on the ground robot's Pi and camera, 10 feet from the piece of paper.

The image below is the Python code, *ImageProcessing.py*, which was used to test the algorithms on the Raspberry Pi. To execute this code:

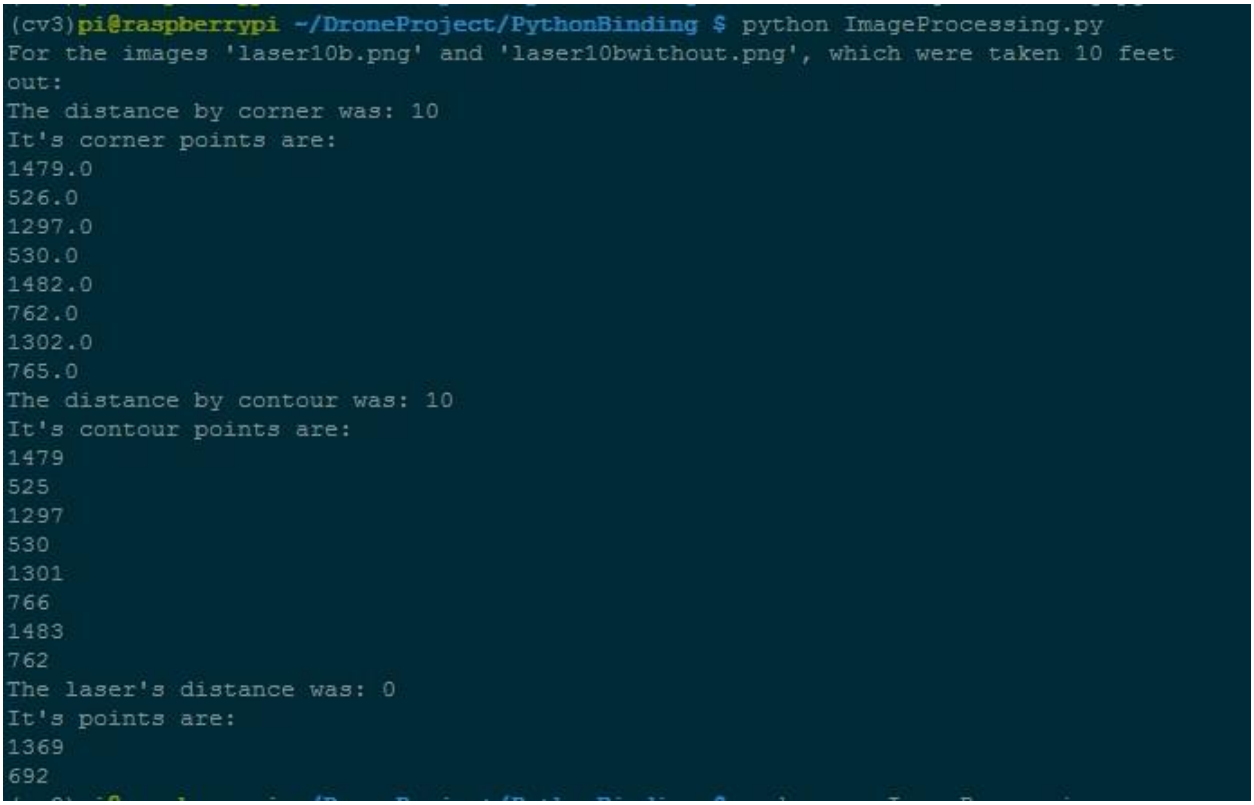
```
$     workon cv3            // Loads the virtual environment
$     python ImageProcessing.py
```

```
#!usr/bin/python
import numpy as np
import time
import cv2
import DroneUtils


image = cv2.imread("laser10b.png");
image2 = cv2.imread("laser10bwithout.png");
print("For the images 'laser10b.png' and 'laser10bwithout.png', which were taken 10 feet out: ");
#Corner methods
distByCorner = DroneUtils.getPaperDistByCorner(image.tostring('c'),image.shape[1],image.shape[0]);
cornerList = DroneUtils.getPaperByCorner(image.tostring('c'), image.shape[1], image.shape[0]);
print("The distance by corner was: %d" % distByCorner);
print("It's corner points are:");
for pt in cornerList:
        print(pt);
#Contour methods
distByContour = DroneUtils.getPaperDistContour(image.tostring('c'), image.shape[1], image.shape[0]);
contourList = DroneUtils.paperContour(image.tostring('c'), image.shape[1], image.shape[0]);
print("The distance by contour was: %d" % distByContour);
print("It's contour points are:");
for pts in contourList:
        print(pts);
#Laser methods
laserDist = DroneUtils.getLaserDist(image.tostring('c'), image2.tostring('c'), image.shape[1], image.shape[0]);
laserList = DroneUtils.getLaserLocation(image.tostring('c'), image2.tostring('c'), image.shape[1], image.shape[0]);
print("The laser's distance was: %d" % laserDist);
print("It's points are:");
for pnt in laserList:
        print(pnt);


cv2.destroyAllWindows()
```

The output of this Python code is shown in the image below:



As expected and shown in the 1st line of the output, the distance found by taking the corner points found in the image in respect to the markers size and ratio was found to be 10 feet.

In the next 9 lines of the output, the corner points found and used in the image are shown. For clarity, the points found are mapped as:

Top – left point: (1297, 530)
Bottom – left point: (1302, 765)
Top – right point: (1479, 526)
Bottom – right point: (1482, 762)

In the 10th line of the output the distance to the marker, found by measuring the contour lines of the paper, was found to be 10 feet consistent with the previously found distance.

The next 9 lines of the output show the corner points based on the contour lines of the paper. These points were consistent with the previously found points. They are mapped as:

Top – left point: (1297, 530)
Bottom – left point: (1301, 766)
Top – right point: (1479, 525)
Bottom – right point: (1483, 762)

The lasers distance was found to be 0, which shows that the algorithm was incorrect and needs to be fixed. However, the point of the laser was accurately found to be (1369, 692), which is near center.

Aiming the Ground Robots Laser

One possible implementation to get the robot to point the laser and camera towards an object in the room is to find the points of the object in an image, and the position of the laser pointer in the same image. Take for example a cardboard box, used in this project to represent an obstacle the robot must navigate around. If the four corner points of this box are known, and the position of the robots laser is also known, then the Raspberry Pi can relay to the Arduino how many degrees to move the servos and in which directions, so that the laser and camera are pointed at the obstacle. This requires a working method for identifying the laser pointers position in an image as well as a working method for identifying the points of the obstacle. The section above on servo calibration addresses in greater detail how to move the servos and laser to the desired position of the obstacle.