Controls Testbed

By: Ezra Idy,
Professor: Vikram Kapila

Date: 9/4/18

Table of Content:

Acknowledgement:

I would first like to thank Professor Vikram Kapila for providing me with the space and resources needed to complete the project. I would also like to thank Sai Prasanth Krishnamoorthy for allowing me to take part and contribute in his project. I would also like to thank him for all the help and advice he has provided me over the span of the project. Lastly I would like to thank my fellow lab mates for the constant support that they provided me.

Abstract:

The aim of this project was to create an affordable testbed that can be used to explain many control concepts, with the use of a motor. Some of these concepts include: PID control, Kalman Filtering (KF), and Sensor Fusion (SF). As robotics is becoming more advanced, many of these concepts are being used in current and future robotics projects. However, due to the complexity of these concepts they are not taught in the educational setting. This project serves as an introduction to these complex topics, using an iOS user interface to analyze and control a continuous 360° servo motor.

1. Introduction:

As robotics is getting more interactive topics such as Sensor Fusion and other advanced control topics are being implemented more in many industrial settings. This is especially true for companies interested in creating autonomous vehicles. Sensor Fusion, or SF, is the gathering of data from multiple sensors in order to gain a more accurate understanding of the environment and the sensors surroundings.

There are three distinct types of SF: Competitive, Cooperative, and Complementary.

Competitive SF: A group of sensors are considered to be competitive if the sensor data is an independent measurement of the same property. Competitive SF is sometimes known as a redundant configuration. This is because the sensors are being used to enhance performance of a given object by reducing the error created form individual sensors.

Cooperative SF: A group of sensors are considered cooperative when two sensors provide independent measurements that help derive information that would not be available from a single sensor. This type of fusion is usually the hardest to design due to the inaccuracies that can be produced. An example of Cooperative SF is when a 3D image can be produced from two 2D images.

Complementary SF: A group of sensors are considered to be complementary if the sensors provide independent data of different objects, and the data is then combined. This type of fusion is generally the easiest to fuse, since the data can be appended to each other. Complementary SF is used to obtain more information about ones surroundings.

The aim of the project is to design a testbed that can stand as an introduction for teaching difficult topic such as: PID control, Kalman Filters, and Sensor Fusion. This testbed acts as a tool to help implement an affordable device that can be used for interactive learning. The sensor fusion being used by the testbed is an example of Competitive SF. Using a camera sensor and a hall effect sensor the angle of a continuous motor can be measured and observed.

2. Hardware

2.1 Parallax Propeller:

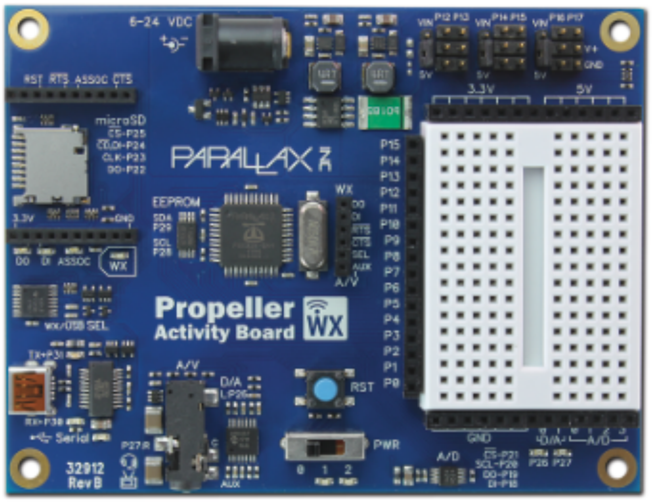


Figure 1: Parallax Propeller Activity Board

The Parallax Propeller Activity Board shown in Figure 1 above is a 8-core multiprocessor microcontroller. The Propeller has 64 KB of EEPROM, and can provide I2C and UART communication. The board has 31 I/O pins, where 15 of them are programmable, operating at an output voltage of 3.3V. The Parallax Propeller board cost around $71.00 USD. For the scope of the project, the Parallax Propeller is used to control the servo motor. The Propeller board sends that information along through serial communication. 5 of the 8 propeller cores were used for this project.

2.2 Raspberry Pi 3:

The Raspberry Pi 3 board, shown in Figure 2 below, is a Quad Core 1.2 GHz Broadcom 64 bit CPU that is the size of a credit card. The board has 40 extended GPIO pins, with 4 USB ports. It has a CSI port for connecting a Raspberry Pi camera. The Raspberry Pi 3 can handle I2C and UART communication as well. The cost of a Raspberry Pi board is around $40.00 USD. For the scope of this project the Raspberry Pi is using the Robotic Operating System, ROS. With ROS the Raspberry Pi can communicate with the iOS user interface, as well as read the camera angle using a Raspberry Pi camera. Using serial communication the Raspberry Pi can communicate with the Parallax Propeller board.

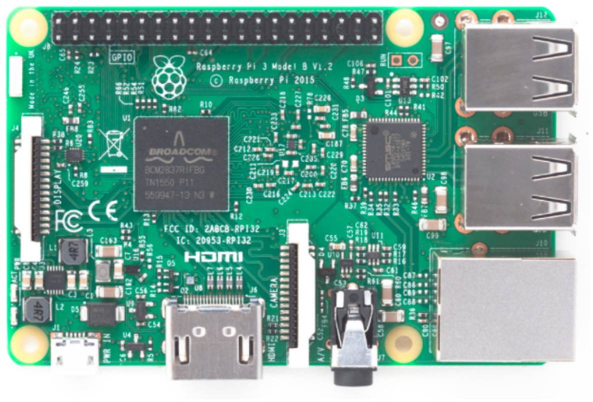


Figure 2: Raspberry Pi 3

2.3 Servo Motor:

Figure 3: Feedback 360° High Speed Servo Motor

The servo motor used for this project is the Parallax Feedback 360° High Speed servo motor. The motor is a bidirectional continuous servo motor with an internal hall effect position sensor. With the use of the hall effect sensor feedback can be provided to the motor and the angles can be held. The motor uses PWM signals for both the feedback pin as well as the control pin. The cost of the motor is around $28.00 USD. The angle is calculated using the duty cycle obtained from the hall effect sensor.

2.4 Camera Sensor:

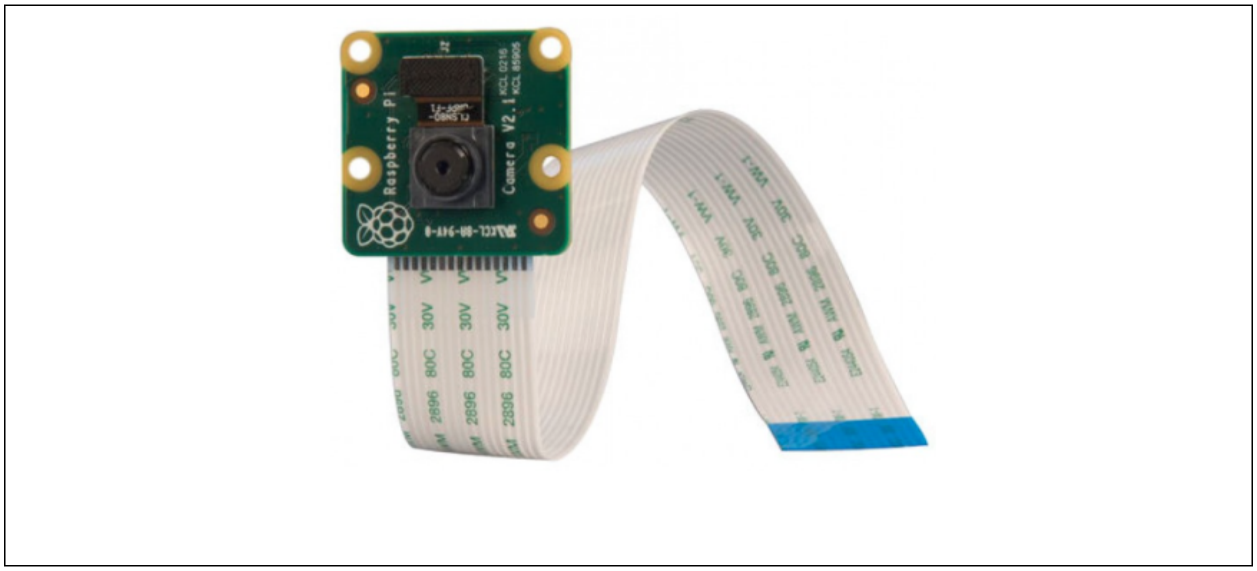


Figure 4: Raspberry Pi camera

A Raspberry Pi camera V2 was used for the vision system for this testbed as shown in Figure 4 above. The image produced by the camera has a resolution of 1280X960. With the camera system the angle of the motor can be calculated using vision. This is done with the use of AprilTags identifiers on the wheel of the motor. Thus, when the camera detects the AprilTag it can determine the orientation of the wheel. Using transformations, one can calculate the angle of the wheel. The cost of the Raspberry Pi camera is around $26.00 USD.

2.5 First Design:

The first design for the testbed consisted of a vision system, two optical encoders, and a standard continuous servo motor. In this design the only controller was a Raspberry Pi, using a UART Servo/PWM Pi HAT shield to control the motor. Using the ticks from the optical encoders, the motor angle and motor direction was able to be determined. This data can then be combined with, and compared to, the data obtained from camera system. The first design of the test bed can be seen in Figure 5. As one can see, the testbed is made to be light and portable.

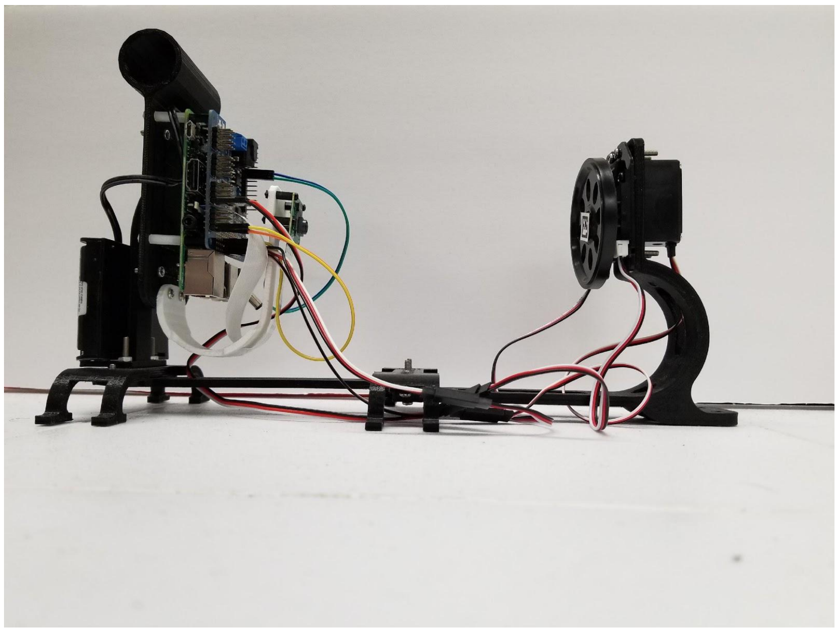


Figure 5: First Testbed Design

2.6 Second Design:

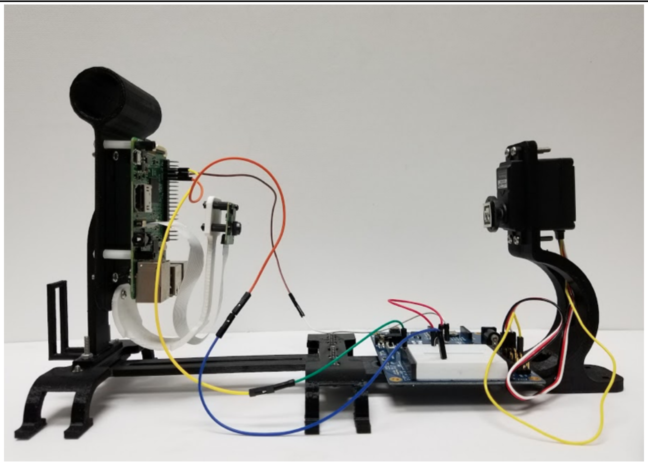


Figure 6: Second Testbed Design

In Figure 6, one can see the second design for the testbed. Using the same 3D printed shell, the testbed is still made to be light and portable. The main difference between the two designs is the use of the Parallax Propeller board. Unlike the first design, the second design uses two microcontrollers to operate. This design change was made in order to work with the Feedback 360° servo motor. When working with this specific servo motor, an input and output PWM pin was needed. After looking at the specs for the Servo/PWM Pi HAT it was clear that it was unable to provide an input pin to obtain the feedback data. This is in part due to the fact that the HAT relies on two pins for UART communication. Trying to keep the system as simple and light as possible, the best solution was to incorporate the Parallax Propeller board into the system. The second design allows for the Propeller board to control the motor. The Propeller board then uses serial communication to communicate with the Raspberry Pi.

2.7 Total Cost:

Table 1: Cost

Parts	Amount	First Design	Second Design
Raspberry Pi 3	1	$40	$40
Parallax Propeller Board	1	$0	$71
Raspberry Pi Camera	1	$26	$26
Servo Motor	1	$15	$28
Servo/PWM Pi HAT	1	$17	$0
Miscellaneous	1	$70	$70
Total	------	$168	$235

From Table 1 above, one can see that the cost of the hardware is relatively cheap for an educational testbed.

3. Software:

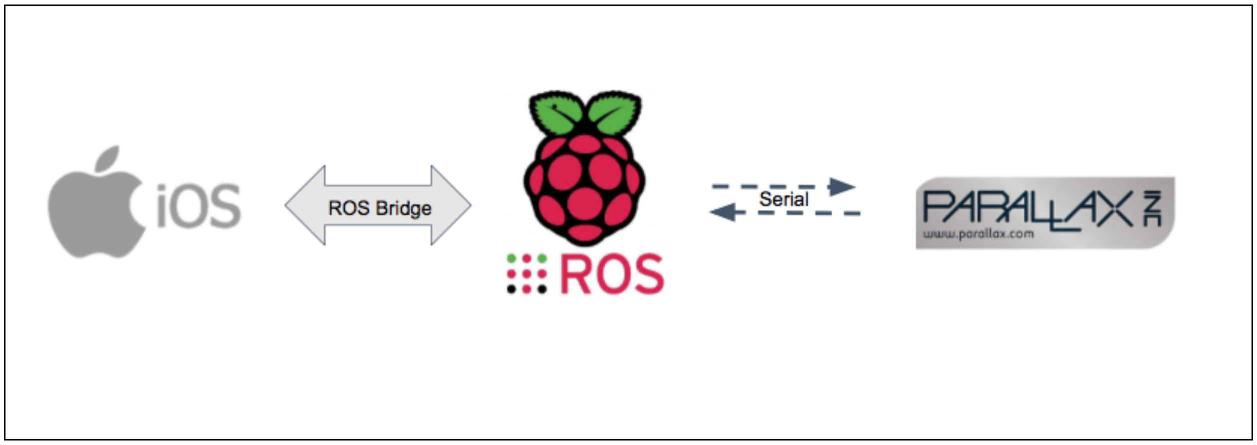
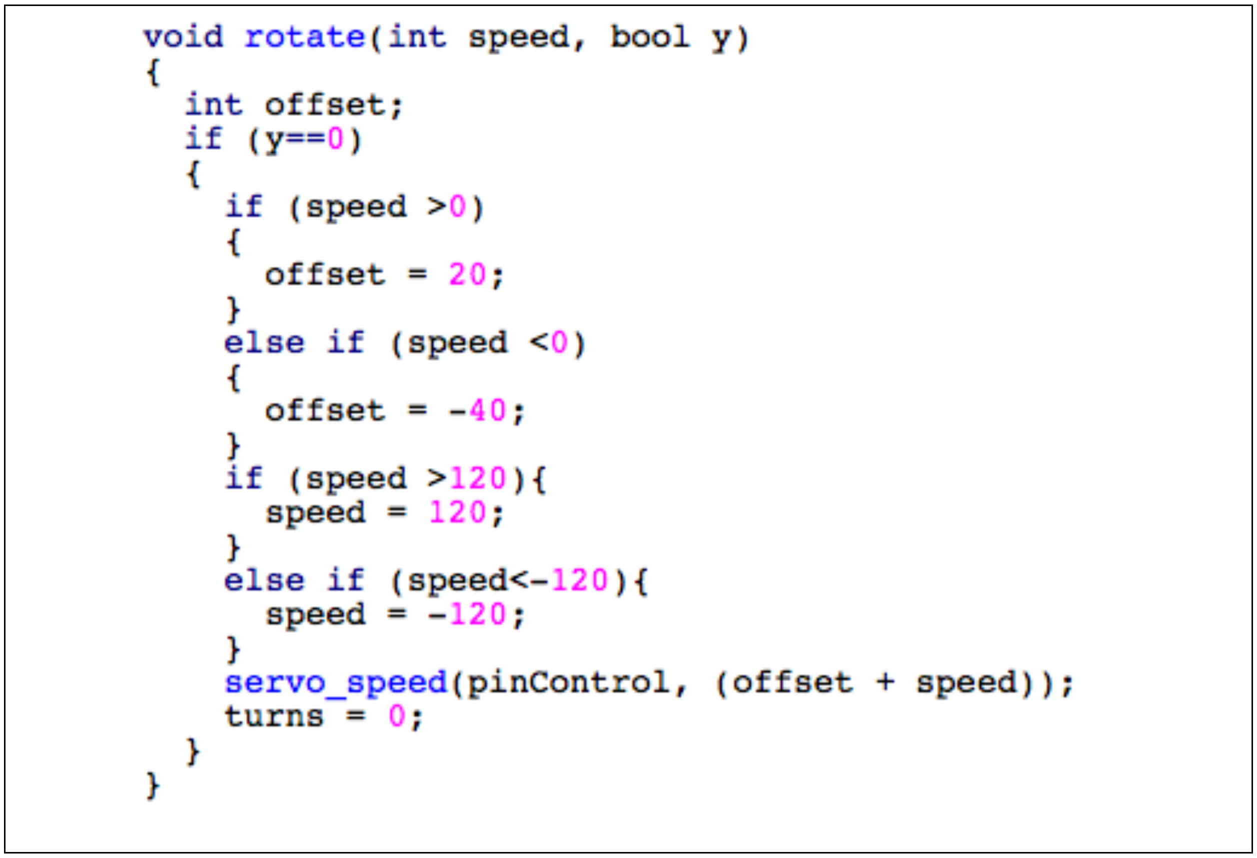


Figure 7: Communication overview

Figure 7 above shows the different types of communication happening within the project. As one can see from Figure 7, the testbed can be broken down into three subsystems: the iOS user interface, the ROS controller, and the Parallax Propeller board. Since the Propeller board is attached to the motor, most of the data processing takes place within one of its 8-cores. The propeller is used to: calculate the fused angle, calculate the Kalman angle, implement the PID control, and to control the speed of the motor. This data is sent to the Raspberry Pi through serial communication. The Raspberry Pi is used to obtain the data of the camera sensor. The Raspberry Pi also acts as a connection between the iOS user interface, and the Propeller board. Thus, the user is able to control the testbed from his/her smart device. The data is then sent over to the Raspberry Pi with the help of ROS Bridge. Once the data is sent to the Raspberry Pi, it is repackaged into a byte array where it is sent through the serial port to the Propeller board. The propeller then unpackages the data and updates all the values that were sent.

3.2 Continuous Rotation:

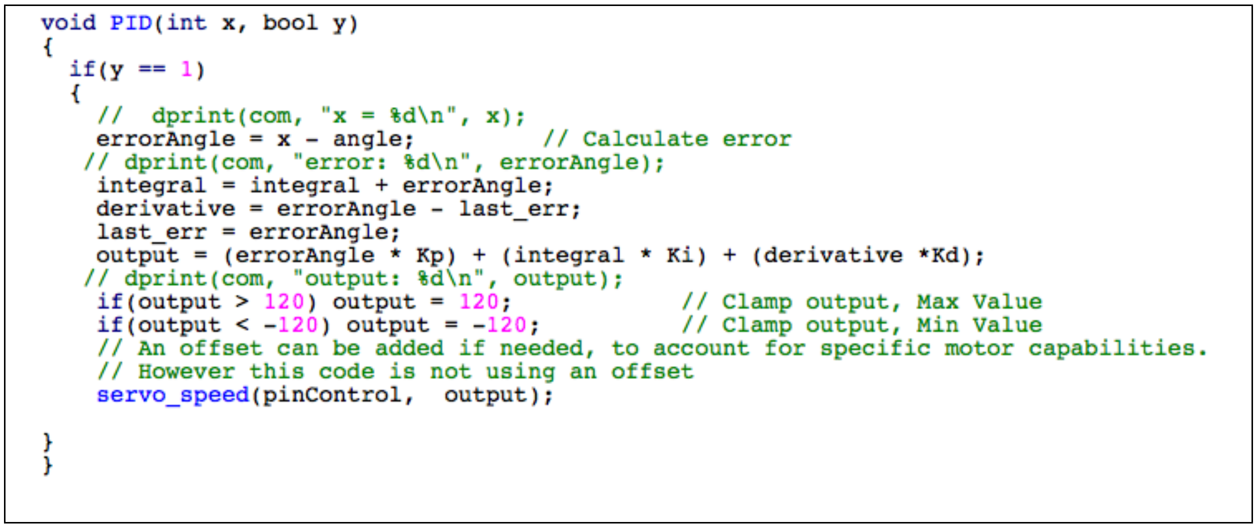


```
void rotate(int speed, bool y)
{
  int offset;
  if (y==0)
  {
    if (speed >0)
    {
      offset = 20;
    }
    else if (speed <0)
    {
      offset = -40;
    }
    if (speed >120){
      speed = 120;
    }
    else if (speed<-120){
      speed = -120;
    }
    servo_speed(pinControl, (offset + speed));
    turns = 0;
  }
}
```

Code 1: Moving servo motor in continuous rotation

Code 1 above is used to move the servo motor in a continuous rotation. In the above code, boolean y is sent from the user interface to control the motor. If y is equal to 0 the motor should run continuously, however if y is equal to 1 the motor should be in the PID control setting shown in Code 2 below. As one can see, speed of the motor is limited to ±120. This was calculated experimentally. If the motor is initially set to a higher speed then 120, the Propeller board is likely to restart automatically. It should also be noted that there is an offset within the motor as well. The offsets were also found through trial and error.

3.3 PID:



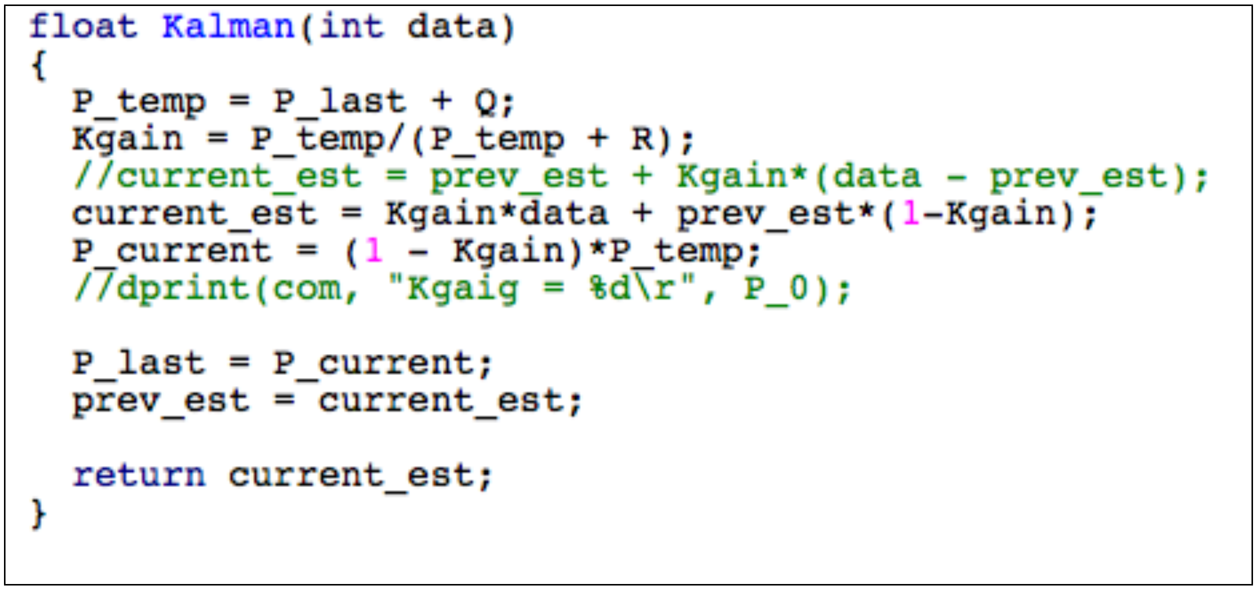
```
void PID(int x, bool y)
{
  if(y == 1)
  {
    //  dprint(com, "x = %d\n", x);
    errorAngle = x - angle;          // Calculate error
   // dprint(com, "error: %d\n", errorAngle);
    integral = integral + errorAngle;
    derivative = errorAngle - last_err;
    last_err = errorAngle;
    output = (errorAngle * Kp) + (integral * Ki) + (derivative *Kd);
   // dprint(com, "output: %d\n", output);
    if(output > 120) output = 120;            // Clamp output, Max Value
    if(output < -120) output = -120;          // Clamp output, Min Value
    // An offset can be added if needed, to account for specific motor capabilities.
    // However this code is not using an offset
    servo_speed(pinControl,  output);

}
}
```

Code 2: Moving servo motor to desired target

The code above shows the PID function created for moving the motor to a specific target angle set by the user. With the user interface, the user can change the values of the PID gains from 0 to 25. The range of the gains is due to the serial communication and the amount of information that can be sent. From the code above, one can see that the output variable has an upper and lower limit. These limits were obtained through trial and error, similar to Code 1 above.

3.4 Kalman Filter:

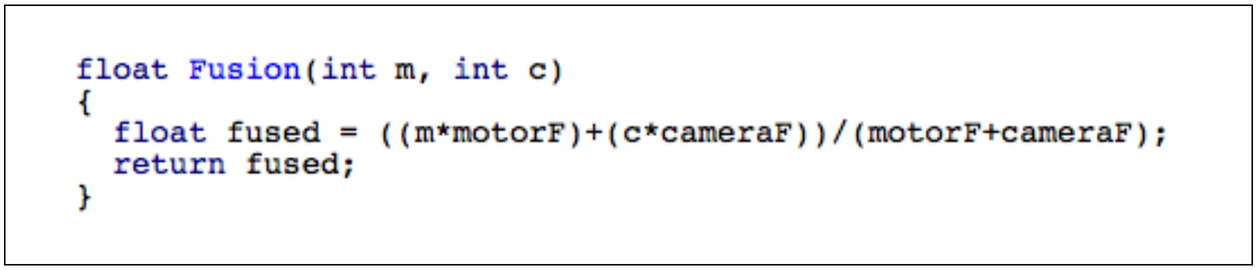
The KF is used to predict the next measurement obtained and reduce the error produced due to noise in the system. The code below shows the Kalman function that was created to demonstrate this concept. The user can choose to turn the KF on or off, however they cannot adjust the noise variables Q and R.



```
float Kalman(int data)
{
  P_temp = P_last + Q;
  Kgain = P_temp/(P_temp + R);
  //current_est = prev_est + Kgain*(data - prev_est);
  current_est = Kgain*data + prev_est*(1-Kgain);
  P_current = (1 - Kgain)*P_temp;
  //dprint(com, "Kgaig = %d\r", P_0);

  P_last = P_current;
  prev_est = current_est;

  return current_est;
}
```

Code 3: Kalman Filter function

3.5 Sensor Fusion:



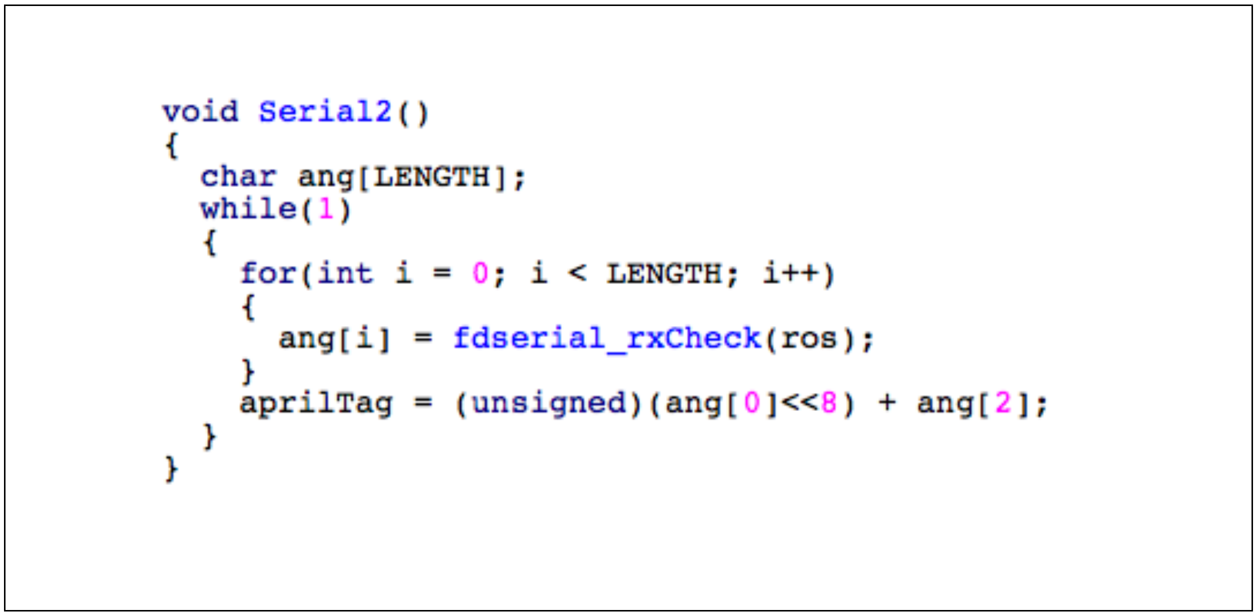
```
float Fusion(int m, int c)
{
  float fused = ((m*motorF)+(c*cameraF))/(motorF+cameraF);
  return fused;
}
```

Code 4: Sensor Fusion function

Code 4 above, shows the algorithm used for the competitive fusion function. The user can change the reliability of the two sensors, and see how each sensor interacts with the movement of the motor. The method of SF used is called voting.

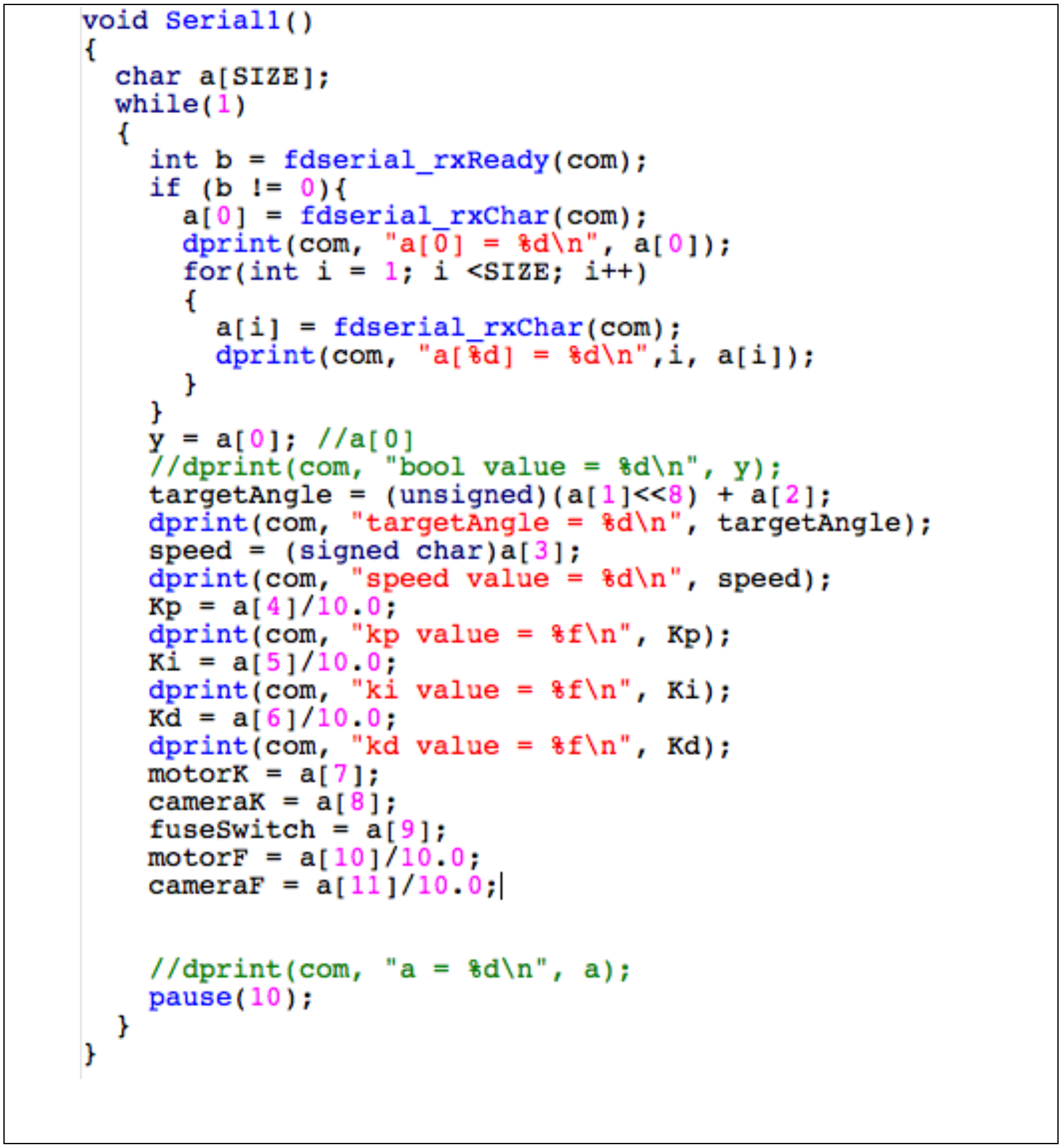
3.6 Serial Communication:

Code 5 and Code 6 below display the serial communication cogs acting on the Parallax Propeller. Both serial cogs are using the fdserial simple_tools library. The reason for two serial communication cogs is because of the way the data is being sent. Since the serial port communicates in bytes, the order in which the bytes are sent is important. The design of the user interface serial communication in Code 7 is meant to only send information if the data is different from the previous information. However, the camera angle is continuously updating. Thus the camera angle data should be sent over through a different serial port. This is done through the second serial cog in Code 6. One can also see that once the bytes are transferred over to the Propeller, they still need to be transformed. This is because some of the values being sent are either greater than 1 byte, ie. target angles or camera angles, or the data has a decimal, ie. Kp, Ki, Kd values.



```
void Serial2()
{
  char ang[LENGTH];
  while(1)
  {
    for(int i = 0; i < LENGTH; i++)
    {
      ang[i] = fdserial_rxCheck(ros);
    }
    aprilTag = (unsigned)(ang[0]<<8) + ang[2];
  }
}
```

Code 5: **Serial communication with camera angle data**



```c
void Serial1()
{
  char a[SIZE];
  while(1)
  {
    int b = fdserial_rxReady(com);
    if (b != 0){
      a[0] = fdserial_rxChar(com);
      dprint(com, "a[0] = %d\n", a[0]);
      for(int i = 1; i <SIZE; i++)
      {
        a[i] = fdserial_rxChar(com);
        dprint(com, "a[%d] = %d\n",i, a[i]);
      }
    }
    y = a[0]; //a[0]
    //dprint(com, "bool value = %d\n", y);
    targetAngle = (unsigned)(a[1]<<8) + a[2];
    dprint(com, "targetAngle = %d\n", targetAngle);
    speed = (signed char)a[3];
    dprint(com, "speed value = %d\n", speed);
    Kp = a[4]/10.0;
    dprint(com, "kp value = %f\n", Kp);
    Ki = a[5]/10.0;
    dprint(com, "ki value = %f\n", Ki);
    Kd = a[6]/10.0;
    dprint(com, "kd value = %f\n", Kd);
    motorK = a[7];
    cameraK = a[8];
    fuseSwitch = a[9];
    motorF = a[10]/10.0;
    cameraF = a[11]/10.0;


    //dprint(com, "a = %d\n", a);
    pause(10);
  }
}
```

Code 6: **Serial communication with user interface data**

4. Issues:

When working on the testbed, some minor issues were faced. One issue that was tackled during the redesign was whether the motor should be controlled by an Arduino Uno, or the Parallax Propeller. The Arduino Uno was an option due to its size, cost, and processing power. However after testing the motor with both controllers, the Propeller was chosen due to its multicores. Another issue that was faced when working on the project was the different communications and data networks. With the testbed, there is a lot of information being transferred between the 3 subsystems and it is important to make sure that the correct information is placed in the right location. Lastly, connecting the iOS device with ROS required a lot of time.

5. Conclusion:

5.1 Future Work:

With the newly designed testbed, more controls are possible. One improvement that can be made is to make the iOS user interface more user friendly. One can also add a graph tab to the user interface to allow for more visualization of what is happening with the motor. With the propeller 8 core multiprocessor, one can introduce more control algorithms. This includes potential interaction with different actuators and sensors. Another improvement that could be made is to reduce the amount of information that is being sent across the three subsystems.

5.2 Conclusion:

The projects goal was achieved and a testbed was created. The testbed is both affordable and mobile, and can be used by any user through the iOS user interface. The testbed is designed to demonstrate and introduce the more complex topics of robotics interested students and teachers. With this testbed, topics such as PID control, Kalman Filtering, and Competitive Sensor Fusion can be looked at more closely and with a real world example.