Lab 1: Set-up and Synchronization of Wallaby with Kiss Web IDE, Simple Motor and Sensor Programs

ECE 564: Fundamentals of Autonomous Robots Lab
Team 1: Jacob Cassady, Chase Crutcher, and Olalekan Olakitan Olowo
August 21, 2019

The group members have worked together and face-to-face at all stages of this project work. The contributions of members to the report and to the codes are equal.
(Initials of group members)

Introduction

The goal of this lab was to practice using the Wallaby Robot Controller with the KIPR Instructional Software System (KISS) Web Integrated Development Environment (IDE). C programs were developed in the KISS IDE to actuate motors using different KISS functions as well as respond to stimuli from light and touch sensors.

Parts

The Wallaby Robot Controller shown in Figure 1 has a Linux based operating system and allows for quick development through a web IP port and a wifi connection.

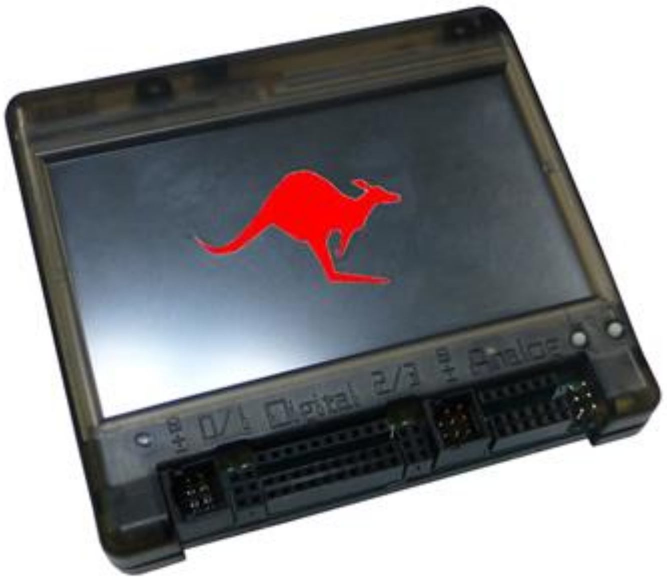


Figure 1: Wallaby Robot Controller

The pin out of the Wallaby Robot Controller is shown in Figure 2. It allows for 4 motor inputs, 10 digital inputs, and 6 analog inputs.

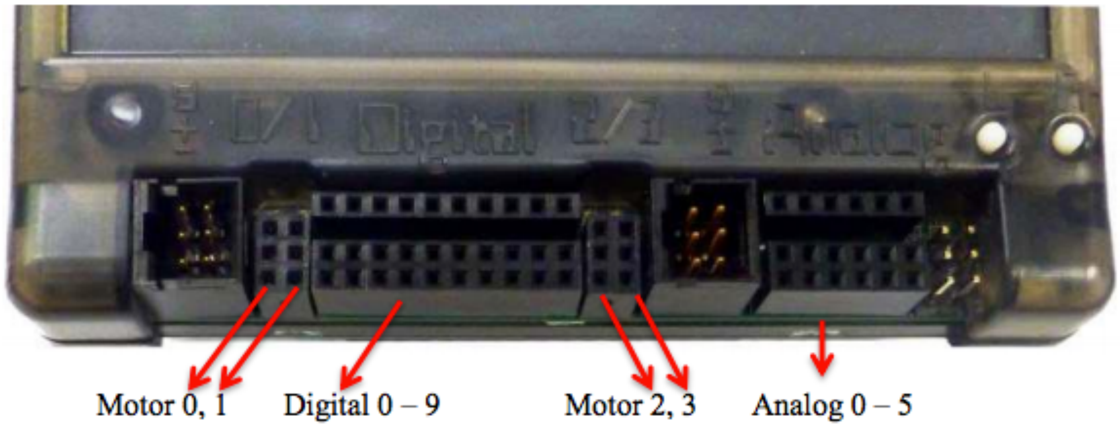


Figure 2: Wallaby Motor Controller Pinout

Figure 3 displays the additional hardware used in this lab. Two motors were used as well as one light sensor and one touch sensor.

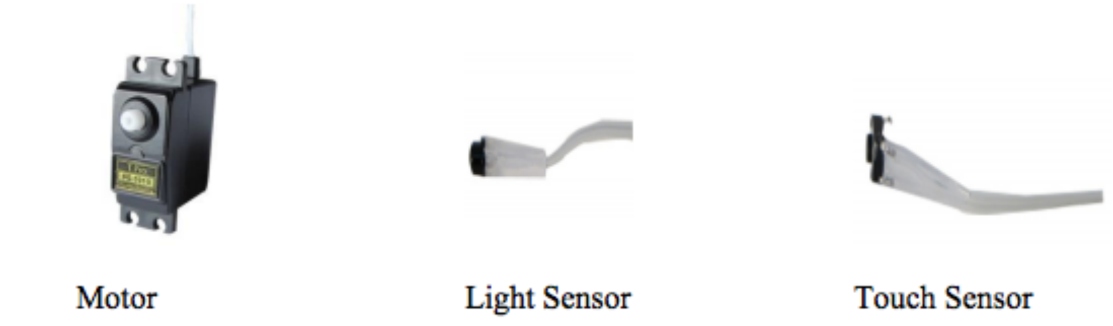
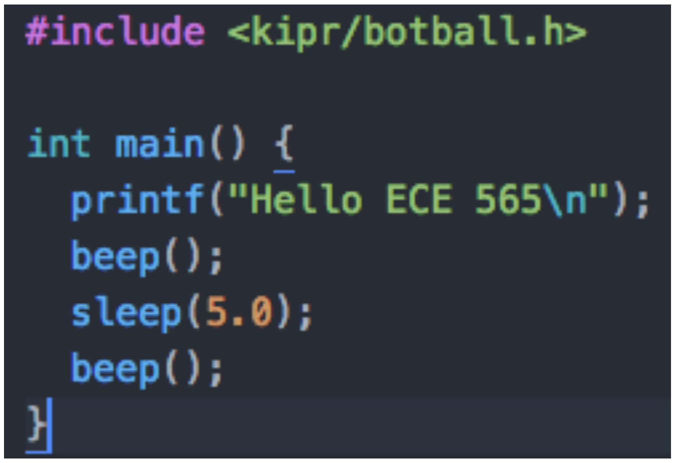


Figure 3: Motor and Sensors

Source Code and Question Responses

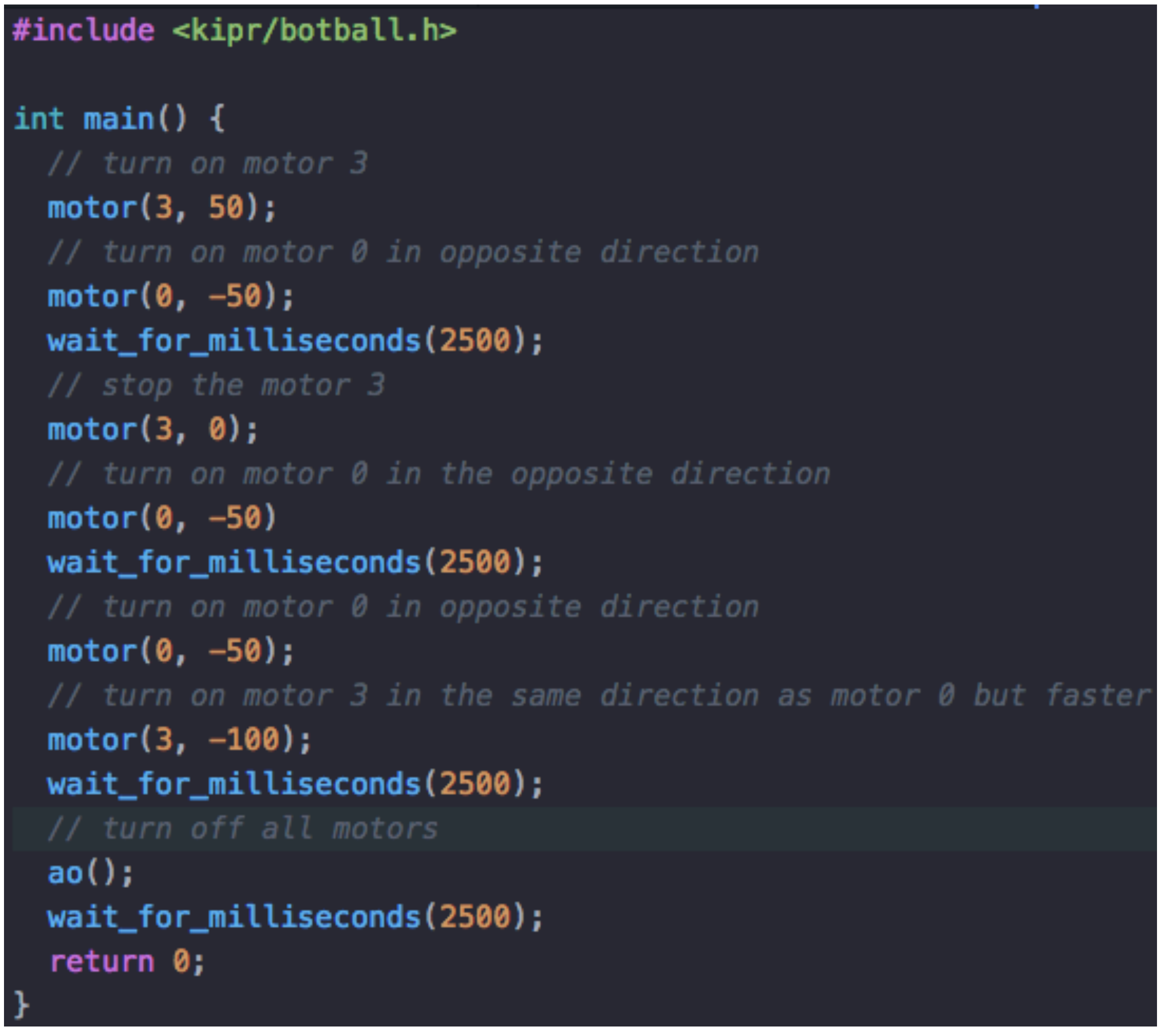
HelloECE.c



```c
#include <kipr/botball.h>

int main() {
  printf("Hello ECE 565\n");
  beep();
  sleep(5.0);
  beep();
}
```

This program prints “Hello ECE 565” on the Wallaby Robot Controller then beeps, waits 5 seconds, and beeps again.

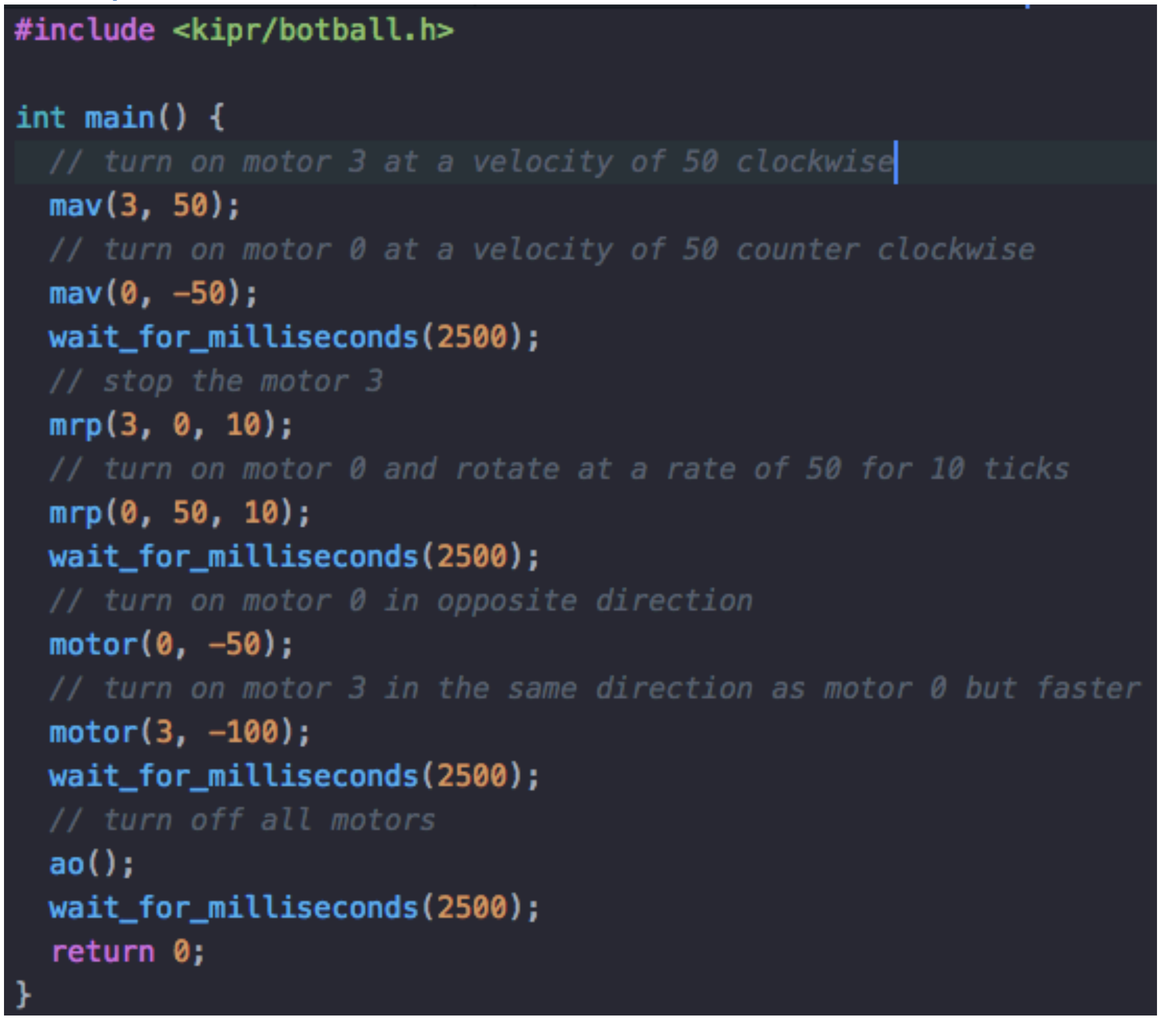
Motorao.c



```
#include <kipr/botball.h>

int main() {
  // turn on motor 3
  motor(3, 50);
  // turn on motor 0 in opposite direction
  motor(0, -50);
  wait_for_milliseconds(2500);
  // stop the motor 3
  motor(3, 0);
  // turn on motor 0 in the opposite direction
  motor(0, -50)
  wait_for_milliseconds(2500);
  // turn on motor 0 in opposite direction
  motor(0, -50);
  // turn on motor 3 in the same direction as motor 0 but faster
  motor(3, -100);
  wait_for_milliseconds(2500);
  // turn off all motors
  ao();
  wait_for_milliseconds(2500);
  return 0;
}
```

The differences between the ao() and motor() functions has to do with the number of motors it will effect. The ao() command will turn off all motors while motor(3,0) would only turn off motor 3.

MavMrp.c



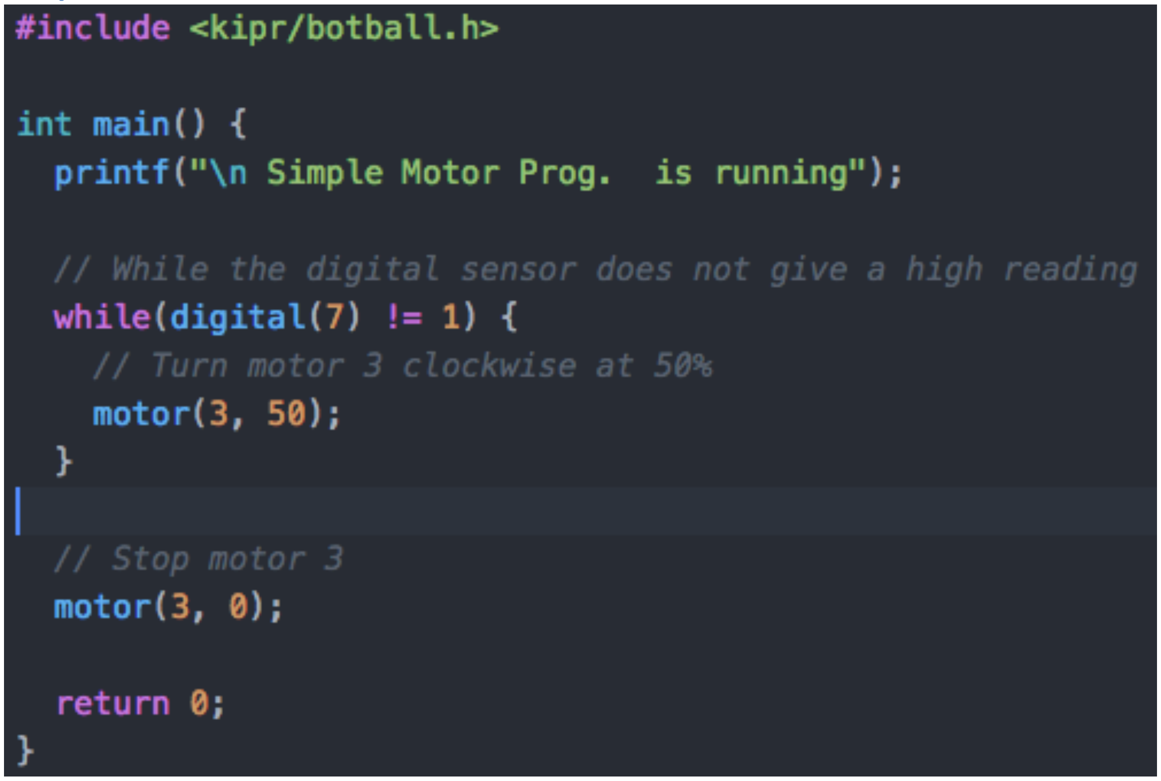
```c
#include <kipr/botball.h>

int main() {
  // turn on motor 3 at a velocity of 50 clockwise
  mav(3, 50);
  // turn on motor 0 at a velocity of 50 counter clockwise
  mav(0, -50);
  wait_for_milliseconds(2500);
  // stop the motor 3
  mrp(3, 0, 10);
  // turn on motor 0 and rotate at a rate of 50 for 10 ticks
  mrp(0, 50, 10);
  wait_for_milliseconds(2500);
  // turn on motor 0 in opposite direction
  motor(0, -50);
  // turn on motor 3 in the same direction as motor 0 but faster
  motor(3, -100);
  wait_for_milliseconds(2500);
  // turn off all motors
  ao();
  wait_for_milliseconds(2500);
  return 0;
}
```

Mav command stands for Move At Velocity. This command takes a motor number and a velocity amount. It is similar to the Motor command, which moves the motor at a percentage of its output instead of a set velocity. On the other hand, the mrp command stands Move Relative Position. This command takes three parameters:

- A motor number
- A velocity
- A number of ticks

It rotates at the velocity for the number of ticks and then stops.

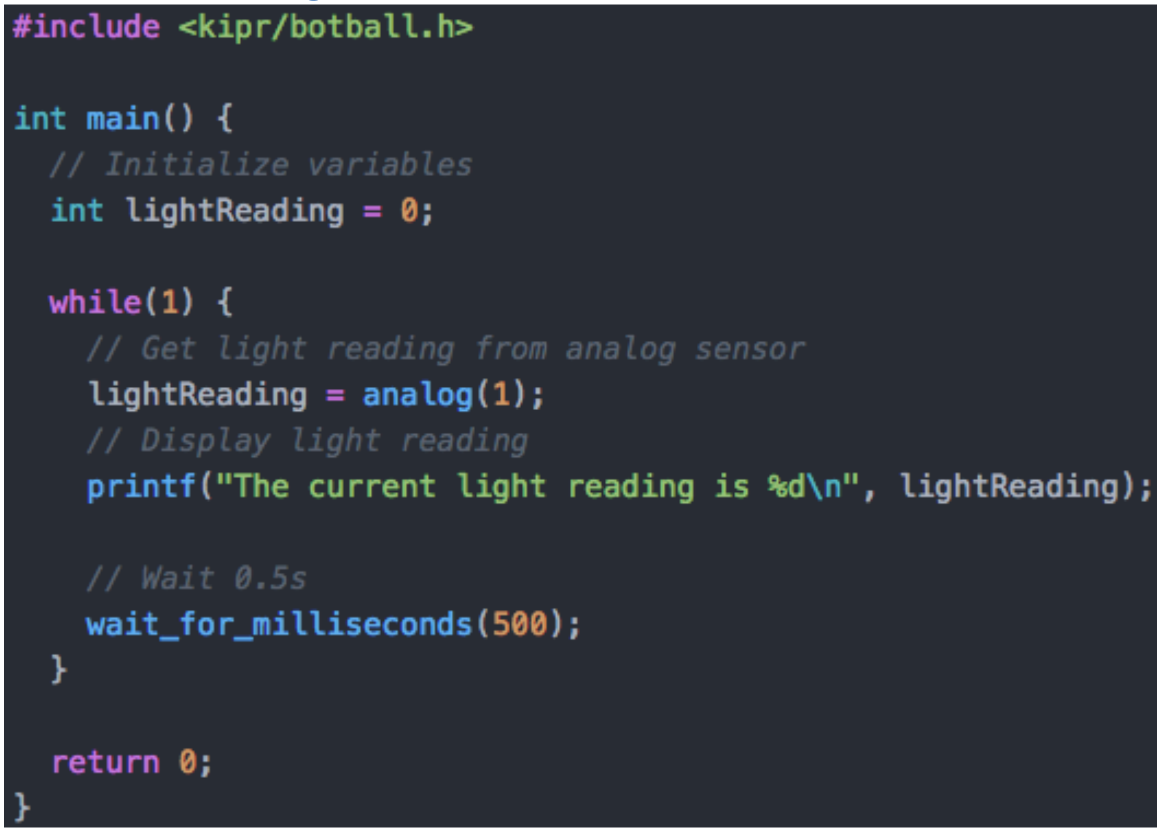
SimpleMotorSwitch.c



```c
#include <kipr/botball.h>

int main() {
  printf("\n Simple Motor Prog.  is running");

  // While the digital sensor does not give a high reading
  while(digital(7) != 1) {
    // Turn motor 3 clockwise at 50%
    motor(3, 50);
  }

  // Stop motor 3
  motor(3, 0);

  return 0;
}
```

This program keeps the motor turning clockwise at a rate of 50% until the digital sensor, in this case a touch switch sensor, receives a stimulus. The motor then turns off and the program exits.

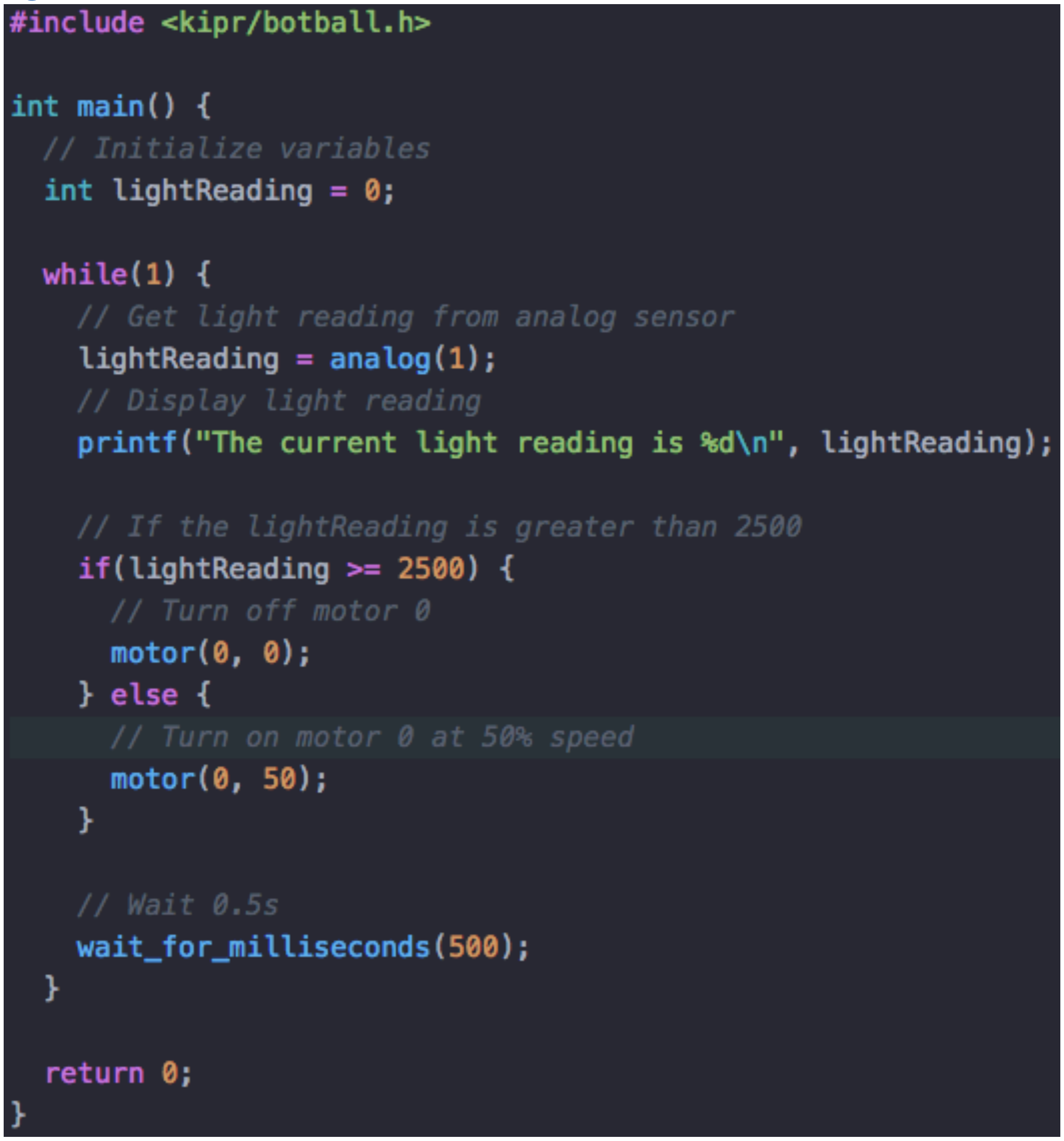
MeasureAmountOfLight.c



```c
#include <kipr/botball.h>

int main() {
  // Initialize variables
  int lightReading = 0;

  while(1) {
    // Get light reading from analog sensor
    lightReading = analog(1);
    // Display light reading
    printf("The current light reading is %d\n", lightReading);

    // Wait 0.5s
    wait_for_milliseconds(500);
  }

  return 0;
}
```

The program above displays the current light reading to the Wallaby Motor Controller's screen. The range for analog sensors is between 0 and 4095. As the amount of light hitting the sensor decrease, the output from the sensor rises in magnitude. On the other hand, as the sensor receives more light the value drops in magnitude. The ambient light sensor reading was around 1000.

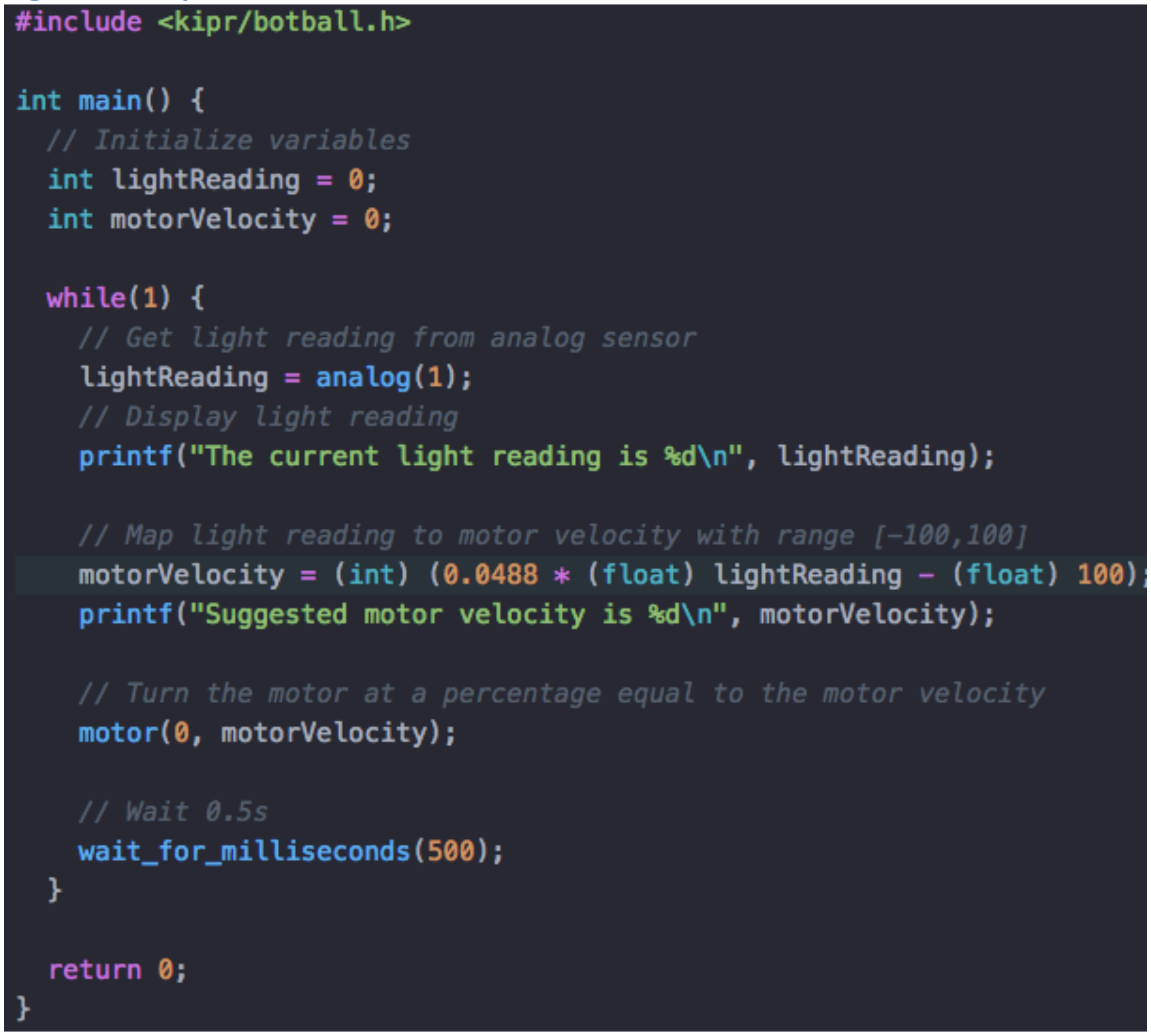
LightMotorSwitch.c



```c
#include <kipr/botball.h>

int main() {
  // Initialize variables
  int lightReading = 0;

  while(1) {
    // Get light reading from analog sensor
    lightReading = analog(1);
    // Display light reading
    printf("The current light reading is %d\n", lightReading);

    // If the lightReading is greater than 2500
    if(lightReading >= 2500) {
      // Turn off motor 0
      motor(0, 0);
    } else {
      // Turn on motor 0 at 50% speed
      motor(0, 50);
    }

    // Wait 0.5s
    wait_for_milliseconds(500);
  }

  return 0;
}
```

This program keeps motor 0 running at 50% until a threshold value of 2500 is received from the light sensor. There is a sweet spot when blocking light where the analog reading will be near 2500, sometimes over and sometimes under. This will make the motor look like it is slowing down before it is completely shut off.

LightMotorSpeed.c



```
#include <kipr/botball.h>

int main() {
  // Initialize variables
  int lightReading = 0;
  int motorVelocity = 0;

  while(1) {
    // Get light reading from analog sensor
    lightReading = analog(1);
    // Display light reading
    printf("The current light reading is %d\n", lightReading);

    // Map light reading to motor velocity with range [-100,100]
    motorVelocity = (int) (0.0488 * (float) lightReading - (float) 100);
    printf("Suggested motor velocity is %d\n", motorVelocity);

    // Turn the motor at a percentage equal to the motor velocity
    motor(0, motorVelocity);

    // Wait 0.5s
    wait_for_milliseconds(500);
  }

  return 0;
}
```

This program receives a light reading, maps it to a motor percentage value between -100 and 100 and drives the motor at this rate. It then waits 0.5 seconds and starts over.

Suggestions

The instructions for updating the firmware may not be pertinent if the wallaby robot controllers are all updated before being used in this lab.