Report 2 - Jakub Hampl & Dénes Findrik

Our second task was to find the docking stations and guide the robot to bumping into the switch, that triggered a sequence of lights to appear.

Hardware modifications

For this task we added two light sensors and a light bulb on the bottom of the robot. We placed the light sensors 2.1 centimeters above the floor and placed them 6.8 centimeters apart. The light bulb is placed in between the light sensors such that light from the light bulb reflects from the floor and is detected by the light sensor. The purpose of this arrangement is to detect the difference between normal lab floor and the darker floor of the docking station (see figure 1).

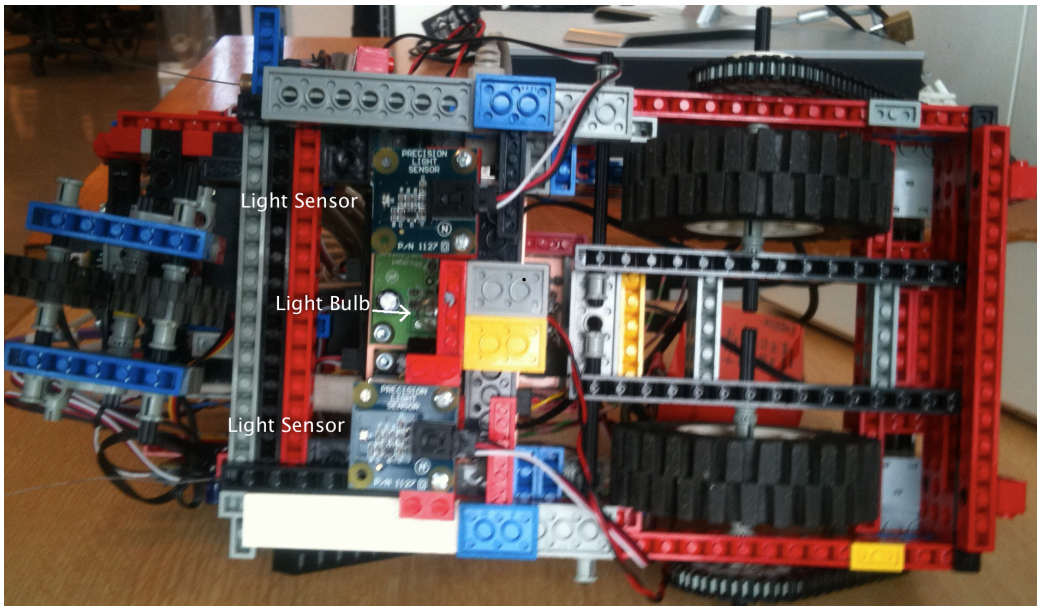

Figure 1: Bottom of the robot showing the placement of the light sensors and the bulb

Furthermore we changed the front wheel from a single wheel to a double wheel construction to lessen pressure on this wheel and have smoother steering (see figure 1 and figure 2 for a bottom and side view of the robot).

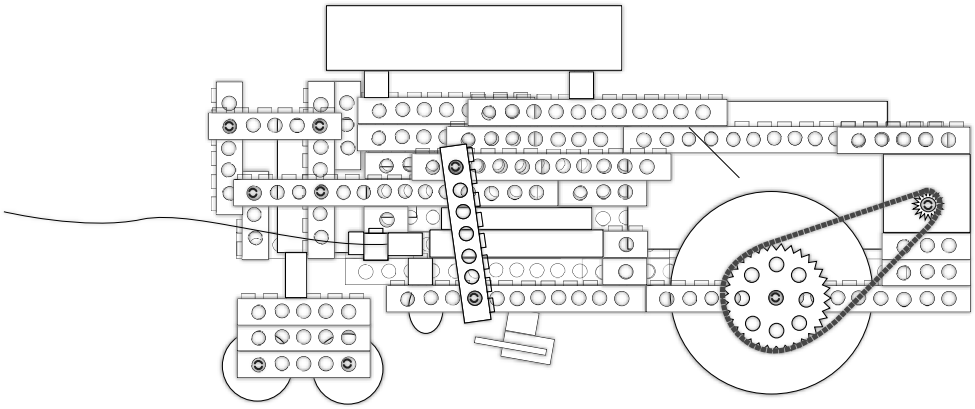

Figure 2: Side view of the robot

Report 2 - Jakub Hampl & Dénes Findrik

Software design

We have kept our reactive architecture reasonably intact from the previous week while simply adding more levels on top. Since we wanted to encourage our robot to explore more of the space, we changed our lowest level behavior from driving in a straight line to following a spiral-like shape. This was implemented using a simple timer as one of the algorithm's inputs.

When the algorithm detects dark flooring underneath, it modifies its reactions to favor whisker input and ignore input from the IR sensor. The whisker does not trigger avoidance anymore but simply slight corrections in course. Here we exploit the shape of the docking stations as these have converging walls thus leading the robot to the center where the switch is. See Figure 3 for a basic overview of the robot's behavior.

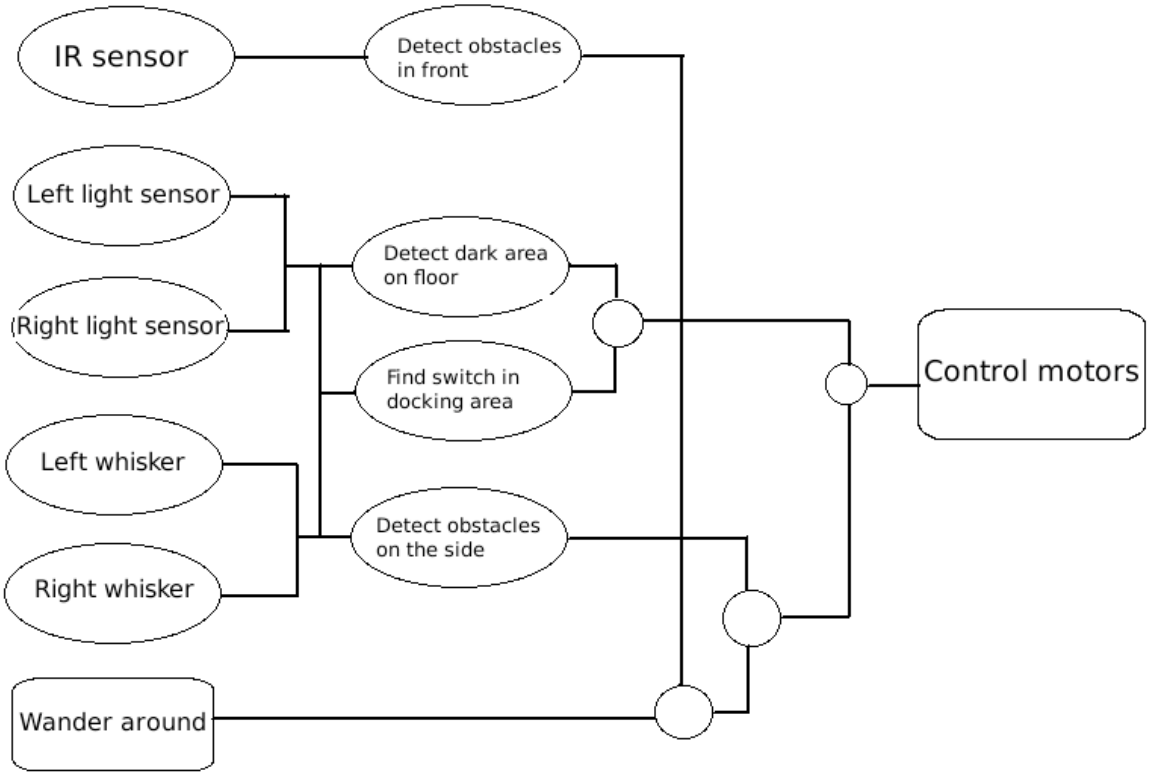

Figure 3: A hierarchical view of the robot's behavior from the inputs to the motor control

We must note at this point that this behavior is still purely reactive and rather primitive, i.e. if the robot per chance hits the wall at an angle contrary to the flow towards the docking station (which is possible in only certain docking stations; most have their internal angle sharp enough to not let this happen), the algorithm will make it leave the docking station without actually triggering the switch. A solution will probably require to add sensing capabilities for the light emitted by the switch and some internal state on whether the switch was triggered or not.

In our present algorithm we assume that we have hit the switch when both whiskers are triggered and we are above dark flooring. This may seem as a bold assumption since when moving perpendicular to the wall this will be triggered (see above for solution). We also slow down to half speed when on dark floor to avoid this danger.

Report 2 - Jakub Hampl & Dénes Findrik

Here is a pseudocode for our control algorithm (see also `Behavior.c`, which implements this):

```
if left light sensor detects dark but not right sensor
    drive slowly forward and to the left
else if right light sensor detects dark but not left sensor
    drive slowly forward and to the right
else if right and left sensors detect dark
    if only left whisker triggered
        drive slowly forward and to the right
    else if only right whisker triggered
        drive slowly forward and to the left
    else if both whiskers triggered
        stop for 2 seconds
        drive back for 3 seconds
        rotate left for 3 seconds
    else
        drive slowly forwards
else if left whisker triggered
    drive backwards to the right for a second
else if right whisker triggered
    drive backwards to the left for a second
else if IR sensor detected something ahead
    drive straight back for a second
    drive backwards to the right for a second
else if timer ticks
    turn and drive left
else
    drive straight
```

Code listing

src/Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <phidget21.h>
#include <unistd.h>
#include <math.h>
#ifndef NO_POWERLIB
#include <libpowerbutton.h>
#endif
#include "Debug.c"
#include "Types.c"
#include "Setup.c"
#include "Movement.c"
#include "Behavior.c"



int main(int argc, char* argv[])
{
  setup();
  orientStraightAndDrive(1);
  #ifdef NO_POWERLIB
  while(1)
  #else
  while(power_button_get_value()<2)
  #endif
  {
    timer.iteration++;
    behave();
    if(timer.iteration == timer.threshold + TURNING_DURATION) {
      timer.iteration = 0;
      timer.threshold += 2;
    }
        sleep(0.1);
  }
  stop();
```

```
  teardown();
  return 0;
}
```

src/Movement.c

```
#define DRIVE_LEFT(Value) CPhidgetMotorControl_setVelocity (motoCon-
trol, 0, -0.75 *Value)
#define DRIVE_RIGHT(Value) CPhidgetMotorControl_setVelocity (motoCon-
trol, 1, 0.75 * Value)
#define SERVO(Value) CPhidgetAdvancedServo_setPosition(servo, 0, Value)

//int firstRetreat = 1;

int stop()
{
  MovementLog("stop()");
  DRIVE_LEFT(0.0);
  DRIVE_RIGHT(0.0);
  return 0;
}

int goTowards(double angle, double percent)
{
  MovementLog("goTowards(%f)", angle);
  // angle must be within 0 and 180 degrees
  // 0 is the servo motor turned fully to the right
  // 180 is the servo motor turned fully to the left
  SERVO(20+(angle*(10/9)));
  //sleep(0.2);
  DRIVE_RIGHT(60*(angle/180)*percent);
  DRIVE_LEFT(60*(1-(angle/180))*percent);
  return 0;
}

int turnOnSpotRight()
{
  MovementLog("turnOnSpotRight()");
```

```
  if(state.ServoPosition != 1)
  {
        //stop();
        SERVO(20);
        //sleep(0.5);
        state.ServoPosition = 1;
  }
  DRIVE_RIGHT(50);
  DRIVE_LEFT(-40);
  return 0;
}


int turnOnSpotLeft()
{
  MovementLog("turnOnSpotLeft()");
  if(state.ServoPosition != -1)
  {
        //stop();
        SERVO(220);
        //sleep(0.5);
        state.ServoPosition = -1;
  }
  DRIVE_RIGHT(-40);
  DRIVE_LEFT(50);
}

int orientStraightAndDrive(double percent)
{
  MovementLog("orientStraightAndDrive()");
  SERVO(120);
  //sleep(0.8);
  DRIVE_RIGHT(60*percent);
  DRIVE_LEFT(60*percent);
  //goTowards(90);
  state.ServoPosition = 0;
  return 0;
```

```
}

int retreat(int right)
{
  MovementLog("retreat(%d)", right);
  if(right==1)
  {
        SERVO(190);
        DRIVE_LEFT(-10);
        DRIVE_RIGHT(-60);
  } else
  {
        SERVO(50);
        DRIVE_LEFT(-60);
        DRIVE_RIGHT(-10);
  }
  //sleep(0.8);
  return 0;
}

int driveBack()
{
  MovementLog("driveBack()");
  SERVO(120);
  DRIVE_RIGHT(-40);
  DRIVE_LEFT(-40);
  sleep(1);
  return 0;
}
```

src/Setup.c

```
int AttachHandler(CPhidgetHandle IFK, void *userptr)
{
  int serialNo;
  const char *name;

  CPhidget_getDeviceName(IFK, &name);
```

```
    CPhidget_getSerialNumber(IFK, &serialNo);

    SetupLog("%s %10d attached!", name, serialNo);

    return 0;
  }

  int DetachHandler(CPhidgetHandle IFK, void *userptr)
  {
    int serialNo;
    const char *name;

    CPhidget_getDeviceName (IFK, &name);
    CPhidget_getSerialNumber(IFK, &serialNo);

    SetupLog("%s %10d detached!", name, serialNo);

    return 0;
  }

  int ErrorHandler(CPhidgetHandle IFK, void *userptr, int ErrorCode,
const char *unknown)
  {
    SetupLog("Error handled. %d - %s", ErrorCode, unknown);
    return 0;
  }

  //callback that will run if an input changes.
  //Index - Index of the input that generated the event, State - boolean
(0 or 1) representing the input state (on or off)
  int IKInputChangeHandler(CPhidgetInterfaceKitHandle IFK, void *usrptr,
int Index, int State)
  {
    switch(Index)
    {
          case 1:
                state.LeftWhisker = State;
        SensorIDebug(state.LeftWhisker);
```

```
                break;
            case 2:
                state.RightWhisker = State;
            SensorIDebug(state.RightWhisker);
                break;
            //case 3:
            //    state.BlackBumper = State;
            //    printf("Black Bumper: %d", State);
            //    break;
            //case 4:
            //    state.RedBumper = State;
            //    printf("Red Bumper: %d", State);
            //    break;
            //case 5:
            //    state.RedPlateBumper = State;
            //    printf("Red plate Bumper: %d", State);
            //    break;
            //case 6:
            //    state.BlueBumper = State;
            //    printf("Blue Bumper: %d", State);
            //    break;
            default:
                SensorLog("Digital Input: %d > State: %d", Index, State);
    }
    return 0;
}

//callback that will run if an output changes.
//Index - Index of the output that generated the event, State - boolean
(0 or 1) representing the output state (on or off)
int IKOutputChangeHandler(CPhidgetInterfaceKitHandle IFK, void *usrptr,
int Index, int State)
{
    printf("Digital Output: %d > State: %d\n", Index, State);
    return 0;
}
```

```
    //callback that will run if the sensor value changes by more than the
OnSensorChange trigger.
    //Index - Index of the sensor that generated the event, Value - the
sensor read value
    int IKSensorChangeHandler(CPhidgetInterfaceKitHandle IFK, void *usrptr,
int Index, int Value)
    {
      switch(Index)
      {
            case 0:
                  SensorLog("Spinning sensor: %d", Value);
                  break;
            case 1:
                  SensorLog("Wheel-attached IR: %d", Value);
                  break;
            case 2:
                  SensorLog("Front-facing IR: %d", Value);
                  state.FrontFacingIR = Value;
                  break;
            case 3:
                  SensorLog("Sonar: %d", Value);
                  break;
            case 4:
                  SensorLog("Right Light sensor: %d", Value);
                        state.RightLight = Value;
                  break;
            case 5:
                  SensorLog("Left Light sensor: %d", Value);
                  state.LeftLight = Value;
                  break;
            case 6:
                  SensorLog("Light sensor: %d", Value);
                  break;
            case 7:
                  SensorLog("Light sensor: %d", Value);
                  break;
      }
      return 0;
```

```
    }


    //Display the properties of the attached phidget to the screen.  We
will be displaying the name, serial number and version of the attached de-
vice.
    //Will also display the number of inputs, outputs, and analog inputs on
the interface kit as well as the state of the ratiometric flag
    //and the current analog sensor sensitivity.
    int IKDisplayProperties(CPhidgetInterfaceKitHandle phid)
    {
      int serialNo, version, numInputs, numOutputs, numSensors, triggerVal,
ratiometric, i;
      const char* ptr;


      CPhidget_getDeviceType((CPhidgetHandle)phid, &ptr);
      CPhidget_getSerialNumber((CPhidgetHandle)phid, &serialNo);
      CPhidget_getDeviceVersion((CPhidgetHandle)phid, &version);


      CPhidgetInterfaceKit_getInputCount(phid, &numInputs);
      CPhidgetInterfaceKit_getOutputCount(phid, &numOutputs);
      CPhidgetInterfaceKit_getSensorCount(phid, &numSensors);
      CPhidgetInterfaceKit_getRatiometric(phid, &ratiometric);


      SetupLog("%s", ptr);
      SetupLog("Serial Number: %10d\nVersion: %8d", serialNo, version);
      SetupLog("# Digital Inputs: %d\n# Digital Outputs: %d", numInputs,
numOutputs);
      SetupLog("# Sensors: %d", numSensors);
      SetupLog("Ratiometric: %d", ratiometric);


      for(i = 0; i < numSensors; i++)
      {
            CPhidgetInterfaceKit_getSensorChangeTrigger (phid, i, &trigger-
Val);


            SetupLog("Sensor#: %d > Sensitivity Trigger: %d\n", i, trigger-
Val);
      }
```

```
      return 0;
    }

    int MCInputChangeHandler(CPhidgetMotorControlHandle MC, void *usrptr,
int Index, int State)
    {
      SetupLog("Input %d > State: %d", Index, State);
      return 0;
    }

    int MCVelocityChangeHandler(CPhidgetMotorControlHandle MC, void
*usrptr, int Index, double Value)
    {
      SetupLog("Motor %d > Current Speed: %f", Index, Value);
      return 0;
    }

    int MCCurrentChangeHandler(CPhidgetMotorControlHandle MC, void *usrptr,
int Index, double Value)
    {
      SetupLog("Motor: %d > Current Draw: %f", Index, Value);
      return 0;
    }

    int MCDisplayProperties(CPhidgetMotorControlHandle phid)
    {
      int serialNo, version, numInputs, numMotors;
      const char* ptr;

      CPhidget_getDeviceType((CPhidgetHandle)phid, &ptr);
      CPhidget_getSerialNumber((CPhidgetHandle)phid, &serialNo);
      CPhidget_getDeviceVersion((CPhidgetHandle)phid, &version);

      CPhidgetMotorControl_getInputCount(phid, &numInputs);
      CPhidgetMotorControl_getMotorCount(phid, &numMotors);

      SetupLog("%s", ptr);
      SetupLog("Serial Number: %10d\nVersion: %8d", serialNo, version);
```

```
    SetupLog("# Inputs: %d\n# Motors: %d", numInputs, numMotors);

    return 0;
}

int ASPositionChangeHandler(CPhidgetAdvancedServoHandle ADVSERVO, void
*usrptr, int Index, double Value)
{
    SetupLog("Motor: %d > Current Position: %f", Index, Value);
    return 0;
}

//Display the properties of the attached phidget to the screen.  We
will be displaying the name, serial number and version of the attached de-
vice.
int ASDisplayProperties(CPhidgetAdvancedServoHandle phid)
{
    int serialNo, version, numMotors;
    const char* ptr;

    CPhidget_getDeviceType((CPhidgetHandle)phid, &ptr);
    CPhidget_getSerialNumber((CPhidgetHandle)phid, &serialNo);
    CPhidget_getDeviceVersion((CPhidgetHandle)phid, &version);

    CPhidgetAdvancedServo_getMotorCount (phid, &numMotors);

    SetupLog("%s", ptr);
    SetupLog("Serial Number: %10d\nVersion: %8d\n# Motors: %d", serialNo,
version, numMotors);

    return 0;
}

CPhidgetInterfaceKitHandle ifKit = 0;
CPhidgetMotorControlHandle motoControl = 0;
CPhidgetAdvancedServoHandle servo = 0;
```

```
    int setup()
    {
      int result, numSensors, i;
      const char *err;
      //handles *Handles;
      //Declare an InterfaceKit handle

      // Setup the IFKit
      CPhidgetInterfaceKit_create(&ifKit);
      CPhidget_set_OnAttach_Handler((CPhidgetHandle)ifKit, AttachHandler,
NULL);
      CPhidget_set_OnDetach_Handler((CPhidgetHandle)ifKit, DetachHandler,
NULL);
      CPhidget_set_OnError_Handler((CPhidgetHandle)ifKit, ErrorHandler,
NULL);
      CPhidgetInterfaceKit_set_OnInputChange_Handler (ifKit, IKInputChange-
Handler, NULL);
      CPhidgetInterfaceKit_set_OnSensorChange_Handler (ifKit, IKSensorChan-
geHandler, NULL);
      CPhidgetInterfaceKit_set_OnOutputChange_Handler (ifKit, IKOutputChan-
geHandler, NULL);
      CPhidget_open((CPhidgetHandle)ifKit, -1);

      //get the program to wait for an interface kit device to be attached
      SetupLog("Waiting for interface kit to be attached....");
      if((result = CPhidget_waitForAttachment((CPhidgetHandle)ifKit,
10000)))
      {
            CPhidget_getErrorDescription(result, &err);
            printf("Problem waiting for attachment: %s\n", err);
            return 0;
      }
      //Display the properties of the attached interface kit device
      IKDisplayProperties(ifKit);

      // Setup motoControl

      CPhidgetMotorControl_create(&motoControl);
      CPhidget_set_OnAttach_Handler((CPhidgetHandle)motoControl, AttachHan-
dler, NULL);
```

```
	CPhidget_set_OnDetach_Handler((CPhidgetHandle)motoControl, DetachHandler, NULL);

	CPhidget_set_OnError_Handler((CPhidgetHandle)motoControl, ErrorHandler, NULL);

	CPhidgetMotorControl_set_OnInputChange_Handler (motoControl, MCInputChangeHandler, NULL);

	CPhidgetMotorControl_set_OnVelocityChange_Handler (motoControl, MCVelocityChangeHandler, NULL);

	CPhidgetMotorControl_set_OnCurrentChange_Handler (motoControl, MCCurrentChangeHandler, NULL);

	CPhidget_open((CPhidgetHandle)motoControl, -1);

	SetupLog("Waiting for MotorControl to be attached....");

	if((result = CPhidget_waitForAttachment((CPhidgetHandle)motoControl, 10000)))

	{

		CPhidget_getErrorDescription(result, &err);

		printf("Problem waiting for attachment: %s\n", err);

		return 0;

	}

	MCDisplayProperties(motoControl);

	CPhidgetMotorControl_setAcceleration (motoControl, 0, 50.00);

	CPhidgetMotorControl_setAcceleration (motoControl, 1, 50.00);


	// Setup AdvancedServo

	CPhidgetAdvancedServo_create(&servo);

	CPhidget_set_OnAttach_Handler((CPhidgetHandle)servo, AttachHandler, NULL);

	CPhidget_set_OnDetach_Handler((CPhidgetHandle)servo, DetachHandler, NULL);

	CPhidget_set_OnError_Handler((CPhidgetHandle)servo, ErrorHandler, NULL);


	CPhidgetAdvancedServo_set_OnPositionChange_Handler(servo, ASPositionChangeHandler, NULL);

	CPhidget_open((CPhidgetHandle)servo, -1);

	SetupLog("Waiting for Phidget to be attached....");

	if((result = CPhidget_waitForAttachment((CPhidgetHandle)servo, 10000)))

	{

		CPhidget_getErrorDescription(result, &err);

		printf("Problem waiting for attachment: %s\n", err);
```

```
        return 0;
    }

    //Display the properties of the attached device
    ASDisplayProperties(servo);
    CPhidgetAdvancedServo_setEngaged(servo, 0, 1);
    state.ServoPosition = 0;
    state.RightWhisker = 0;
    state.LeftWhisker = 0;
    state.FrontFacingIR = 0;
    timer.threshold = 10;
    timer.iteration = 0;

    #ifndef NO_POWERLIB
    power_button_reset();

    while(power_button_get_value()==0)
    {
        sleep(1);
    }
    #endif

    return 0;
}

int teardown()
{
    SetupLog("Closing...\n");
    #ifndef NO_POWERLIB
    power_button_reset();
    #endif


    CPhidgetAdvancedServo_setPosition(servo, 0, 120);
    sleep(2);
    // Close IFKIT
    CPhidget_close((CPhidgetHandle)ifKit);
```

```
  CPhidget_delete((CPhidgetHandle)ifKit);
  // close motoControl
        CPhidget_close((CPhidgetHandle)motoControl);
  CPhidget_delete((CPhidgetHandle)motoControl);
  // disengage and close servo
  CPhidgetAdvancedServo_setEngaged(servo, 0, 0);
  CPhidget_close((CPhidgetHandle)servo);
  CPhidget_delete((CPhidgetHandle)servo);
  //all done, exit
  return 0;
}
```

src/Types.c

```
struct stateT {
  unsigned int ServoPosition;
  unsigned int LeftWhisker;
  unsigned int RightWhisker;
  int FrontFacingIR;
        int LeftLight;
        int RightLight;
} state;

struct timerT {
  int iteration;
  int threshold;
} timer;
```

src/Behavior.c

```
#define TURNING_DURATION 4
#define LIGHT_THRESHOLD 160
void behave() {
  //BehaviorIDebug(state.LeftLight);
  //BehaviorIDebug(state.RightLight);
  if(state.LeftLight > LIGHT_THRESHOLD && state.RightLight <
LIGHT_THRESHOLD) {
```

```
        goTowards(20,0.5);
        BehaviorLog("Left light triggered");
      }
      else if (state.LeftLight < LIGHT_THRESHOLD && state.RightLight >
LIGHT_THRESHOLD) {
        goTowards(160,0.5);
        BehaviorLog("Right light triggered");
      }
      else if (state.LeftLight > LIGHT_THRESHOLD && state.RightLight >
LIGHT_THRESHOLD) {

        if(state.LeftWhisker && state.RightWhisker == 0)  {
          BehaviorLog("Both light and left whisker");
          goTowards(80,0.5);
        } else if(state.RightWhisker && state.LeftWhisker == 0)  {
          BehaviorLog("Both light and right whisker");
              goTowards(100,0.5);
        } else if(state.RightWhisker && state.LeftWhisker)  {
          BehaviorLog("Both light and both whiskers");
              stop();
              sleep(2);
              driveBack();
              sleep(3);
              turnOnSpotLeft();
              sleep(3);
        } else {
          BehaviorLog("Both light and no whiskers");
          orientStraightAndDrive(0.5);
        }
      }
       else if(state.LeftWhisker)  {
       BehaviorLog("Left whisker triggered");
       retreat(0);
       sleep(1);
       driveBack();
      }
      else if(state.RightWhisker) {
```

```
        BehaviorLog("Reft whisker triggered");
        retreat(1);
        sleep(1);
        driveBack();
      }
      else if(state.FrontFacingIR > 350) {
        BehaviorLog("IR triggered (%d)", state.FrontFacingIR);
        driveBack();
        retreat(1);
        sleep(1);
        driveBack();
      }
      else if(timer.iteration > timer.threshold && timer.iteration <
timer.threshold + TURNING_DURATION) {
        BehaviorLog("Turning %d", timer.threshold + TURNING_DURATION -
timer.iteration);
        goTowards(30,1);
      }
      else {
            orientStraightAndDrive(1);
      }
    }
```

src/Debug.c

```
    #define Log(...) printf(__VA_ARGS__); printf("\n");
    #define IDebug(arg) printf("arg = %d\n", arg);

    #ifdef SHOULD_DEBUG_BEHAVIOR
    #define BehaviorLog(...) printf(__VA_ARGS__); printf("\n");
    #define BehaviorIDebug(arg) printf("arg = %d\n", arg);
    #else
    #define BehaviorLog(...)
    #define BehaviorIDebug(arg)
    #endif

    #ifdef SHOULD_DEBUG_MOVEMENT
    #define MovementLog(...) printf(__VA_ARGS__); printf("\n");
```

```
#define MovementIDebug(arg) printf("arg = %d\n", arg)
#else
#define MovementLog(...)
#define MovementIDebug(arg)
#endif

#ifdef SHOULD_DEBUG_SETUP
#define SetupLog(...) printf(__VA_ARGS__); printf("\n");
#define SetupIDebug(arg) printf("arg = %d\n", arg);
#else
#define SetupLog(...)
#define SetupIDebug(arg)
#endif

#ifndef SHOULD_DEBUG_SENSOR
#define SensorLog(...)
#define SensorIDebug(arg)
#else
#define SensorLog(...) printf(__VA_ARGS__); printf("\n");
#define SensorIDebug(arg) printf("arg = %d\n", arg);
#endif
```