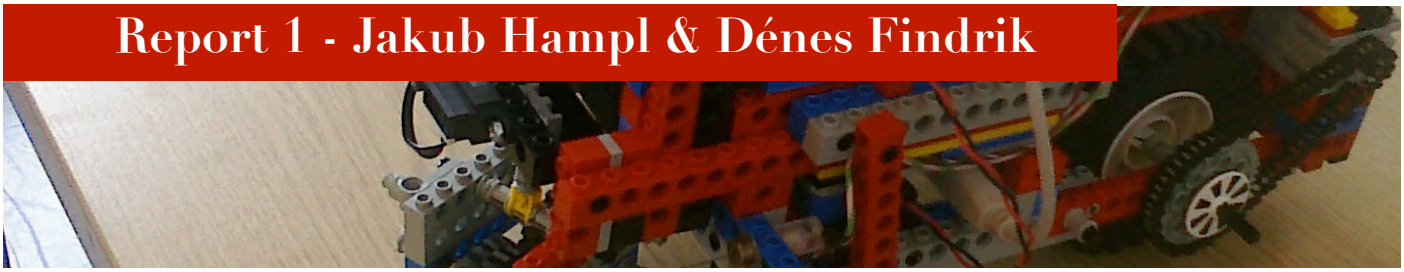
Report 1 - Jakub Hampl & Dénes Findrik



In the first two weeks we concentrated primarily on the construction of a robust frame for the robot that would support our chosen method of locomotion. For the first task (obstacle avoidance) we used a simple reactive architecture.

Building the robot

Our first decision was about locomotion. How to steer efficiently? We decided to use a three wheel design, with the back two wheels each having an independent motor and the front wheel using a servo to set its orientation. Due to the irregular build of the servo motor, we had trouble attaching the front wheel in a robust enough matter. However, after a couple of iterations we managed to attach it properly and also redistribute weight such that the front wheel is now very stable.

Another problem discovered while testing was that the wheels did not have sufficient traction to move the robot forward. This was solved using appropriate gearing and by moving the wheels closer to the center of mass of the robot.

Control and obstacle avoidance

Our control algorithm is very simple. The robot has three sensors (that is, only three are currently used): a front facing IR sensor and two whiskers facing forward with about 35° angles to the right and left. This spread is used such that this covers the entire front profile of the robot. Thus the presumption is that if neither the IR sensors or either of the whiskers are activated then the immediate area in front of the robot is clear. This assumption seemed to be validated in our testing in the robot arena where the objects have such shape and orientation that this holds. However, when testing elsewhere in the lab, our robot tends to get stuck on low or very thin objects that manage to evade the specific configuration of the sensors. We are thinking of adding a large bumper covering the area to mitigate this problem.

Thus the basic behavior of our robot is to drive forward. If one of the sensors is triggered, the robot will instead drive backwards while simultaneously turning in a different direction (this is generally away from the direction of stimulation, e.g. when the left whisker is triggered the robot will try to steer to the right, see Figure 1). The IR has an arbitrarily chosen direction in which the robot turns. The whiskers are given priority over the IR sensor due to the fact that the IR sensor has greater range and thus should trigger earlier than the whiskers.

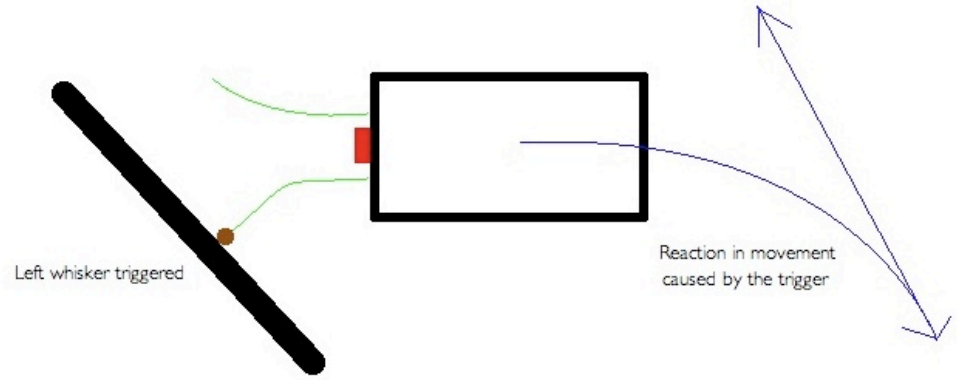


Figure 1: The reaction to the left whisker being triggered

Evaluation and further work

As previously mentioned the robot seems to use these primitive behaviors to avoid common obstacles. However, further parts of the task may require to either change these primitive behaviors or to create an overarching architecture that will prioritize other behaviors to meet the project goals.

Regarding the physical construction of the robot further challenges may include placing more sensors and creating robust frames for them. Also managing all the connecting cables has been a problem in our current design with cables often interfering with correct functioning of sensors.

Code Listing

src/Main.c

```
#include <stdio.h>
#include <phidget21.h>
#include <unistd.h>
#include <math.h>
#include "Types.c"
#include "Setup.c"
#include "Movement.c"

#include <libpowerbutton.h>


int main(int argc, char* argv[])
{
   setup();
   power_button_reset();

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

   orientStraightAndDrive();

   while(power_button_get_value()<2)
   {
          printf("state.ServoPositon = %d", state.ServoPosition);
          if(state.LeftWhisker)  {
                 retreat(0);
                 sleep(1);
                 driveBack();
          }
          else if(state.RightWhisker) {
                 retreat(1);
                 sleep(1);
                 driveBack();
          }
          else if(state.FrontFacingIR > 350) {
                 driveBack();
                 retreat(1);
                 sleep(1);
                 driveBack();
          }
          else {
                 orientStraightAndDrive();
          }
          sleep(0.2);
   }
   power_button_reset();
   stop();

   teardown();
   return 0;
}
```

src/Movement.c

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

    //int firstRetreat = 1;

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

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


    int goTowards(double 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_LEFT(60*(angle/180));
       DRIVE_RIGHT(60*(1-(angle/180)));
       return 0;
    }

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

    int orientStraightAndDrive()
    {
       SERVO(120);
       //sleep(0.8);
       DRIVE_RIGHT(60);
```

```
   DRIVE_LEFT(60);
   //goTowards(90);
   state.ServoPosition = 0;
   return 0;
}

int retreat(int 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()
{
   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);

       printf("%s %10d attached!\n", name, serialNo);

       return 0;
    }

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

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

       printf("%s %10d detached!\n", name, serialNo);

       return 0;
    }

    int ErrorHandler(CPhidgetHandle IFK, void *userptr, int ErrorCode, const char
*unknown)
    {
       printf("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 In-
dex, int State)
    {
       switch(Index)
       {
              case 1:
                     state.LeftWhisker = State;
                     printf("Left Whisker: %d", State);
                     break;
              case 2:
                     state.RightWhisker = State;
                     printf("Right Whisker: %d", State);
                     break;
              default:
                     printf("Digital Input: %d > State: %d\n", 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 In-
dex, 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 OnSensor-
Change trigger.
    //Index - Index of the sensor that generated the event, Value - the sensor read
value
    int IKSensorChangeHandler(CPhidgetInterfaceKitHandle IFK, void *usrptr, int In-
dex, int Value)
    {
       switch(Index)
       {
              case 0:
                     printf("Spinning sensor: %d", Value);
                     break;
              case 1:
                     printf("Wheel-attached IR: %d", Value);
                     break;
              case 2:
                     printf("Front-facing IR: %d", Value);
                     state.FrontFacingIR = Value;
                     break;
              case 3:
                     printf("Sonar: %d", Value);
                     break;
              case 4:
                     printf("Light sensor: %d", Value);
                     break;
              case 5:
                     printf("Light sensor: %d", Value);
                     break;
              case 6:
                     printf("Light sensor: %d", Value);
                     break;
              case 7:
                     printf("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 device.
    //Will also display the number of inputs, outputs, and analog inputs on the in-
terface 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, rati-
ometric, 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);

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

	for(i = 0; i < numSensors; i++)
	{
		CPhidgetInterfaceKit_getSensorChangeTrigger (phid, i, &triggerVal);

		printf("Sensor#: %d > Sensitivity Trigger: %d\n", i, triggerVal);
	}

	return 0;
    }


    int MCInputChangeHandler(CPhidgetMotorControlHandle MC, void *usrptr, int In-
dex, int State)
    {
	printf("Input %d > State: %d\n", Index, State);
	return 0;
    }

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

    int MCCurrentChangeHandler(CPhidgetMotorControlHandle MC, void *usrptr, int In-
dex, double Value)
    {
	printf("Motor: %d > Current Draw: %f\n", 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);

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

	return 0;
    }
```

```
    int ASPositionChangeHandler(CPhidgetAdvancedServoHandle ADVSERVO, void *usrptr,
int Index, double Value)
    {
       printf("Motor: %d > Current Position: %f\n", 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 device.
    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);

       printf("%s\n", ptr);
       printf("Serial Number: %10d\nVersion: %8d\n# Motors: %d\n", serialNo, ver-
sion, 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, IKInputChangeHandler,
NULL);
       CPhidgetInterfaceKit_set_OnSensorChange_Handler (ifKit, IKSensorChangeHan-
dler, NULL);
       CPhidgetInterfaceKit_set_OnOutputChange_Handler (ifKit, IKOutputChangeHan-
dler, NULL);
       CPhidget_open((CPhidgetHandle)ifKit, -1);

       //get the program to wait for an interface kit device to be attached
       printf("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, AttachHandler,
NULL);
	CPhidget_set_OnDetach_Handler((CPhidgetHandle)motoControl, DetachHandler,
NULL);
	CPhidget_set_OnError_Handler((CPhidgetHandle)motoControl, ErrorHandler,
NULL);
	CPhidgetMotorControl_set_OnInputChange_Handler (motoControl, MCInputChange-
Handler, NULL);
	CPhidgetMotorControl_set_OnVelocityChange_Handler (motoControl, MCVeloci-
tyChangeHandler, NULL);
	CPhidgetMotorControl_set_OnCurrentChange_Handler (motoControl, MCCurrentChan-
geHandler, NULL);
	CPhidget_open((CPhidgetHandle)motoControl, -1);
	printf("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, ASPositionChange-
Handler, NULL);
	CPhidget_open((CPhidgetHandle)servo, -1);
	printf("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;
	return 0;
}

int teardown()
{
  printf("Closing...\n");
```

```
    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;
} state;
```