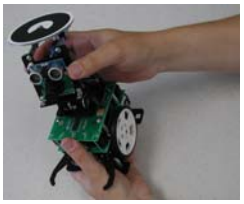
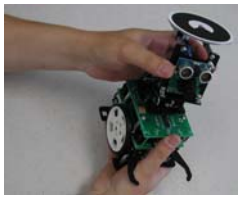


COMP4807 - LAB #1
Behavior-Based Programming
(Due Sun. Sept. 30, 2012 @ 11:59pm)



In this assignment you will become familiar with the process for writing, compiling and running code on the robot. You will gain experience in using the servos, IR sensors, block sensor, beeper and camera. You will also get a chance to implement some simple behaviors (i.e., collision avoidance, wandering and block pushing) and monitor the robot using the **RobotTracker** software (which will be used for all LABS in the course).

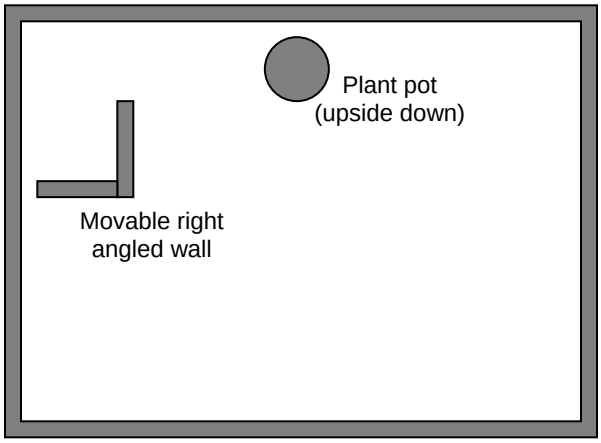
Lab Machine Setup:

Since this is your first lab, information about how to get things working will be described in this section. You will need to remember this for all further lab assignments. When you arrive at the lab, take your designated lab seating. It actually doesn't matter which position you take, as long as you stay at that position all term. At your workstation, there should be a robot and a USB cable with an adapter on the end. Please leave the USB cable plugged into the USB port at all times. There is also a webcam (suspended over the work area) plugged into your computer at the back. Please leave this plugged in. On rare occasion, the camera may stop working...if this occurs, unplug the camera at the back USB port, wait 5 seconds and then plug it back in again. Also, there is a little blue Bluetooth device plugged into the back of the computer... leave this plugged in at all times, or you will not be able to communicate with the robot. The lab environment contains a 5'x7' working area with some simple obstacles. Please configure the robot roaming area according to how you are instructed for the particular lab you are doing. Do NOT attempt to move the fixed (i.e., screwed together) wooden boundaries at any time.

There in a single account on the machines in the labs (COMP4807 Student), so everyone has access to everything. Please DO NOT install any software on these machines and DO NOT modify the screen saver, desktop background or anything at all. Since many others will use the same machine as you, make sure that you remove your code once you are done. DO NOT modify the **Robot Tracker** folder in any way. Backup all your code on a flash drive or zip it and email it to yourself at the end of each lab session. There is no time to repeat experiments if you lose your code and test results. The desktop contains the RobotTracker application. The **course notes** and **sample code** are available on the course webpage when you need to get them.

Environment Setup:

For this lab, you should set up the roaming environment roughly as shown here (to the right). You will need the movable right-angled wall portion and the plant pot placed upside down. Remove and place aside anything else in the area. Make sure that you see the setup shown here when you look at the RobotTracker tracking images to get the correct orientation.

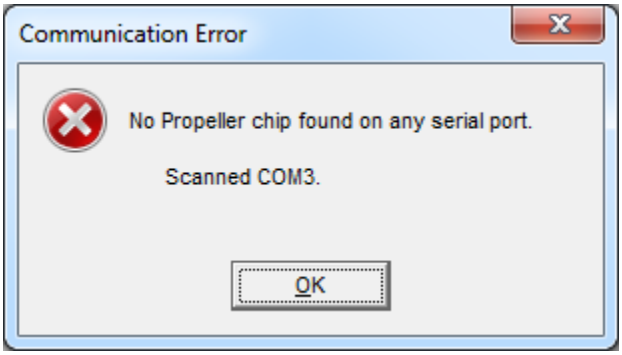


Robot Setup and Calibration:

1. Make sure the robot is turned **off** and turn **on** (i.e., down position) dip switch **1** (**F_IR**) from the 8-position switch on the top of the robot and turn **off** (i.e., up) all other switches (i.e., **2, 3, 4, 5, 6, 7** and **8**). At the back of the robot, make sure that the 2-position dipswitch is set so that both switches are off (i.e., up).

2. **!!! GENTLY !!!** plug the USB cable into the robot. Note that the USB connector is simply soldered at the end and may come off over time if everyone is not gentle with it … which means the robot becomes unusable and you cannot complete your lab.

3. Download and open the **ServoCalibration.spin** program from the course website to start the Parallax Tool IDE. Select **Identify Hardware…** from the **Run** menu. If the robot is off, you should see a similar message to that shown here (but the COM port number will be different).



4. Turn on the robot and servos (i.e., set robot's on switch to position 1) and then press **F11** in the Parallax Tool IDE to compile and download the code onto the robot. The program should download and start. You will hear a startup beeping sound to tell you that the program started.

5. Refer to slides 100 and 101 in chapter 3 of the notes to calibrate the servos. See slides 75 and 76 for where to use these values in your program. The **leftServoStoppedValue** and **rightServoStoppedValue** will be used in your program. The values vary from robot to robot, so you may want to test your servo speeds once in a while throughout the term to determine the stop values just in case they have drifted over time or if you change robots. In addition to the wheel servos, you will want to calibrate the servos for the grippers. Store the values in your program as the following constants (although values will vary between robots):

```
LEFT_GRIPPER_CLOSED = 215
LEFT_GRIPPER_STRAIGHT = 173
LEFT_GRIPPER_WIDEST = 147
RIGHT_GRIPPER_CLOSED = 104
RIGHT_GRIPPER_STRAIGHT = 147
```

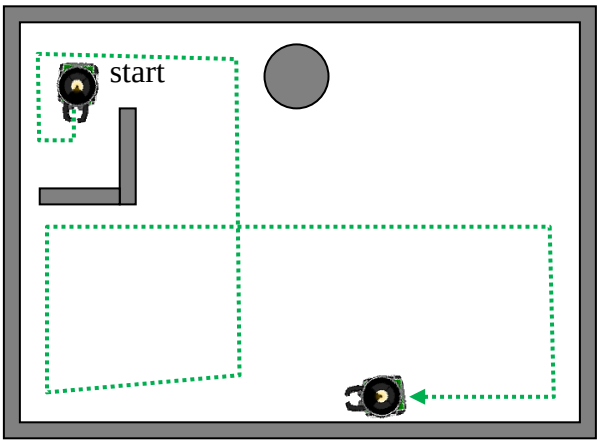
```
RIGHT_GRIPPER_WIDEST = 163
PITCH_DOWNMOST = 95
PITCH_HORIZONTAL = 143
PITCH_UPMOST = 175
YAW_LEFTMOST = 60
YAW_STRAIGHT = 148
YAW_RIGHTMOST = 215
```

Be careful when determining the DOWNMOST and UPMOST positions so that the robot's head is not too close to the top board of the robot. The head will wobble a bit as the robot moves around, so make sure to leave enough of a gap between the head and the robot's top board. Note also that due to the gearing ratio and mounting of the servos, it may not be possible to determine left and right values that are 180° apart. So, assuming that 0° is facing straight ahead, you may be able to choose left and right values that allow the robot to face -85° and +95° but will likely not be able to get the robot to face -90° and +90°.

The Fun Stuff:

6. Write a new spin program, called **ForwardHalt.spin** that makes the robot move forward upon startup and then when one of its 3 front IR sensors detects an obstacle, it should emit a beep from the beeper and stop. Use different beeping frequencies for each IR sensor. When the IR sensor no longer detects an obstacle the robot should move forward again. Try using your hand or foot to halt the robot. You will need the **IR8SensorArray.spin**, and **Beeper.spin** and **ServoControl.spin** object files. Save your code … you will submit it.

7. Write a spin program called **AvoidCollisions.spin** that implements a collision avoidance behavior so that the robot moves straight until it detects an obstacle. If the front left IR sensor detects an obstacle, the robot should turn right until the sensor no longer detects an obstacle on its left. Similarly the robot should turn left when the front right IR sensor detects an obstacle. If all three front sensors detect an obstacle, then select a random choice of moving either left or right.

   Make sure that your robot does not get "stuck" in a corner at any time. Even if it is able to get out after a few tries ... this is not good enough. If you cannot solve this problem, leave it and continue to the next question so that you don't waste time.

   Use **ramping** to allow the robot to accelerate and decelerate as it moves. Start the **RobotTracker** and make sure that your robot is being tracked properly (see chapter 3 notes). By default, the tracker is set to track at around **10** frames per second. For final testing, capture a video clip of the robot moving by using the movie button.

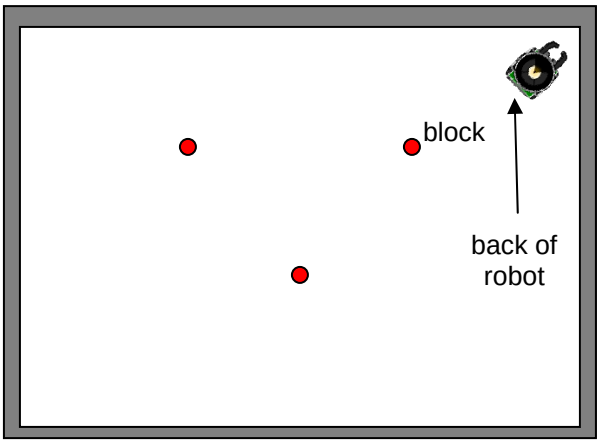
   For the movie, make a new trace and run a 3-minute test of the robot in which it shows how the robot is able begin at the start position (shown above) and produce a path similar to what is shown. Once done, stop the trace and the tracker. Take a screen snapshot of the

RobotTracker window with the trace showing (i.e., select **Show Actual Path** from the **View** menu) ... save it as a **.png** file called **AvoidCollisions.png**. Make sure that the image is shown in the snapshot so that the environmental setup is clearly discernable. Make sure to set a path color that is clearly visible with high contrast when compared to the background. Hand in this file as well as the **.avi** (called **AvoidCollisions.avi**) of the robot moving.

8. Write a program called **Wander.spin** in which you add a wandering behavior to your previous working collision avoidance code (with ramping) so that the robot occasionally turns randomly when it is moving unobstructed in the environment. Adjust your code so that the robot never travels straight but instead always moves in arcs of various radii. Make sure that the collision avoidance has higher priority. The choice of which direction to turn (right or left) as well as the amount to turn (e.g., 5° to 90°) should also be determined randomly. Do not worry about computing the actual number of degrees that the robot turns, simply allow for various amounts. Also, the robot should not randomly turn too often, but it should be noticeable when the robot is placed down in the environment for a while. Perhaps it should turn at least once every 5 seconds, but make this a random decision. Obtain another video clip (saved as **Wander.avi**) and snapshot file (called **Wander.png**) of the robot's movements over a 3 minute period of wandering in the environment while avoiding collisions as before. Again, make sure that the image is shown in the snapshot so that the environmental setup is clearly discernable. Make sure to set a path color that is clearly visible with high contrast when compared to the background.

9. Remove the right angled wall piece as well as the flower pot from the environment and set them aside. At the back of the robot, turn on dipswitch **1** (**Block**). On the top of the robot, turn on dipswitch **5** (**CAM**).

10. Create a program called **BlockDeliver.spin** in which you add a block pushing behavior to your previous working wander code so that it *grabs* any cylindrical block that it finds and brings it to a wall and then backs up, leaving it there. You will need to use the **BlockSensor.spin** object file to detect when you have found a block. Also, make sure that you grab the block with the grippers, using your calibrated values determined earlier. Put the robot in front of the block to test your behavior. Once your code works, modify it so that it uses the camera (i.e., make use of the **CMUCam.spin** object) to search for the blocks instead of wandering randomly. Your code should look for a specific color of block. From the course website, download and make use of the program called **CameraColorSampler.spin** to get the color values for the particular color of block that you are looking for. As you will notice, the color values will fluctuate quite a bit. The program should find exactly 3 blocks, drop them off, and then continue wandering. It will be tricky finding a way to ignore the blocks that you already pushed to the walls … but do your best. Note that the robot should wander around seemingly aimlessly when it does not see any blocks in its current camera view. You may want to have the robot tip its head forward while searching for blocks and lift it when travelling with the block. As before, run a 3 minute trace in which you place the blocks as shown in the picture here. Note that the robot should start facing a corner of the environment and that it does not matter which order the blocks are found in. Make sure to take a snapshot of the environment before you start the trace. Make sure that the image is



shown in the snapshot so that the environmental setup is clearly discernable. Make sure to set a path color that is clearly visible with high contrast when compared to the background. Make a video clip of your robot working. Save your video (**BlockDeliver.avi**) and snapshot file (**BlockDeliver.png**).

Submission:

Create a webpage for this course that contains a simple report on your lab (i.e., a description of what you did and what each part of the assignment was trying to do). Include downloadable links for all of your code as well as any snapshots, trace files and videos. There is no need to make the webpage beautiful … but that's up to you. Login to Carleton's WebCT system and submit a link to your website. Make sure that the website is up and running at least until your code is marked. If you have never created an HTML page before, you can use Mozilla's **Sea Monkey** browser which has a built in editor that allows you to make simple pages. Always keep a backup of all your work (perhaps on a USB key or burn a CD). Here is a summary of what to hand in:

SPIN CODE:

- `ForwardHalt.spin`
- `AvoidCollisions.spin`
- `Wander.spin`
- `BlockDeliver.spin`

SCREEN SNAPSHOTS:

- `AvoidCollisions.png`
- `Wander.png`
- `BlockDeliver.png`

VIDEO CLIPS:

- `AvoidCollisions.avi`
- `Wander.avi`
- `BlockDeliver.avi`