ENGR3390: Fundamentals of Robotics Tutorial (THINKLab - behaviors, arbitration, OCU) 2020a

Pretty much all robot control software shares a SENSE-THINK-ACT data flow in some manner or another. In fancier academic language “perception” feeds into “cognition” which commands “actuation”. You will find deep resources in background material; technical papers, books, videos, journal articles, in all three of these areas as you move into the robotics technical space. A robotics-engineer can build a whole career investigating and building new technology in just one of these areas.

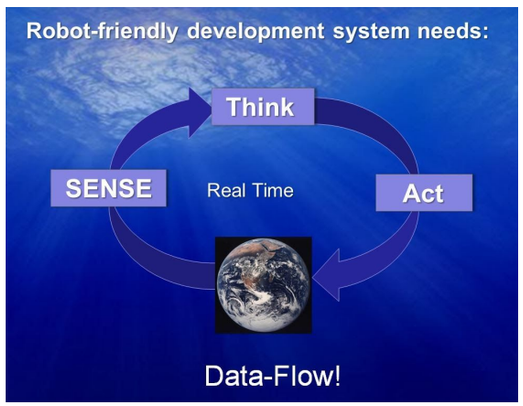
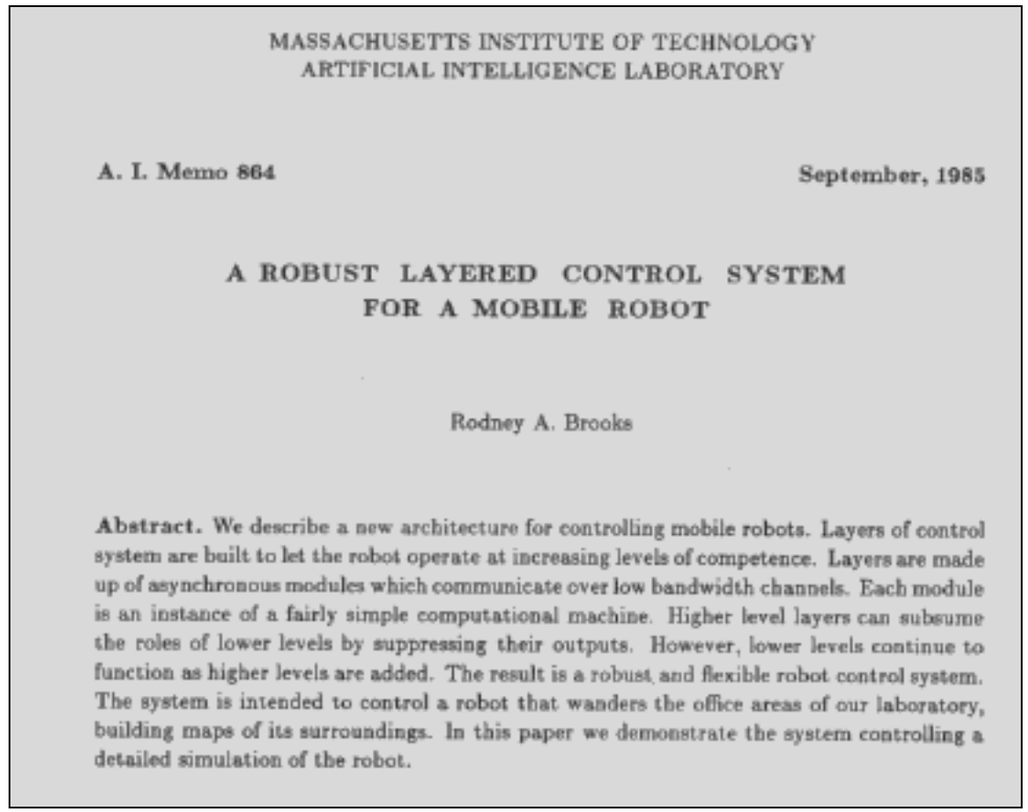


Figure : Sense-Think-Act Control Loop

THINK: involves designing robot behaviors {follow wall, avoid obstacle, move toward target, pick up container, follow lane, stop for pedestrian, weld on joint line, etc.} that receive data from the robot sensor suite processing code {SENSE: “Where am I?”, “What is around me?” and “How am I?”}, perform useful mission based cognition on this input data stream {based on sensor data, have robot do something useful}, runs the outputs from the THINK behaviors into a set of arbiters which finally command the robots actuators (ACT: send setpoints to the actuator code to execute).

That's a pretty big wordy sentence. In practice, for this lab and the Arduino tools you have already used, you will be given a set of SENSE functions, you will create a both handful of THINK behavior functions, and one or two THINK arbiter functions (a arbiter for each main actuation system; wheels, arms, pan/tilt, etc. is pretty standard) and will use a small set of pre-written ACT actuator functions. You can then just add those new functions to the Arduino SENSE-THINK-ACT robot control template or the Laptop one that you developed in the first tutorial for this course.

Let us look at the theory behind all of this a bit before diving right into the mechanics of how to code it. It's important to understand why you are doing something, before you master how you do it. Many of today’s robotics engineers feel that that all modern robotics started with Prof. Rod Brook’s seminal paper on behavior based robotics:



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo 864 September, 1985

A ROBUST LAYERED CONTROL SYSTEM
FOR A MOBILE ROBOT

Rodney A. Brooks

Abstract. We describe a new architecture for controlling mobile robots. Layers of control system are built to let the robot operate at increasing levels of competence. Layers are made up of asynchronous modules which communicate over low bandwidth channels. Each module is an instance of a fairly simple computational machine. Higher level layers can subsume the roles of lower levels by suppressing their outputs. However, lower levels continue to function as higher levels are added. The result is a robust and flexible robot control system. The system is intended to control a robot that wanders the office areas of our laboratory, building maps of its surroundings. In this paper we demonstrate the system controlling a detailed simulation of the robot.

Figure: Brooks, Behaviour based control

ENGR3390: Fundamentals of Robotics Tutorial (THINKLab - behaviors, arbitration, OCU) 2020a

I was lucky enough to work for him in the old days at the MIT Artificial Intelligence Lab where this work started. Please see our course folder for full text of paper and give it a read through. This paper (and the steady stream of papers out of his MIT Mobot lab) changed the course of modern robotics. Before Brooks, robots were slow, dumb and required enormous rooms full of computers to laboriously build fragile simulation “toy block’ world models of the real world in order to make even the smallest decision. After Brooks, robot cars autonomously drive at high speed through Boston on a daily basis.

The original Brooks robots were of about the same level of complexity as those we seek to build in this Fun-Robot class, so following in the footsteps of a successful strategy, we will be building a simplified version of his **Subsumption** architecture, using multiple robot **Behaviors** simultaneously feeding commands into a set of simple **Arbiters** to let your Olin robots perform complex multi-modal missions, in real-time, in a complex real world. In a brief summary, before Brooks, robot control software was written as a linear sequential procedure, one, long, block of dense robot control code:

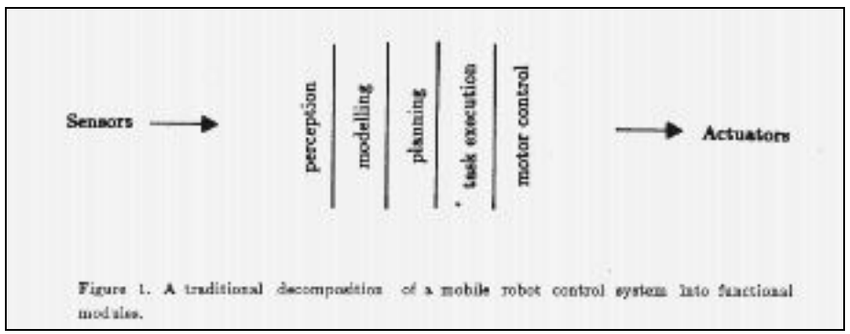


Figure 1. A traditional decomposition of a mobile robot control system into functional modules.

Brook’s key insight, was that biological systems don’t run that way. Instead their control is a complex tapestry of many behaviors, running in parallel, where higher order behaviors can “subsume” control over actuators if needed, but the lower level behaviors will always robustly control the actuators unless/until a higher behavior steps in. Your breath control is a great example. Take a deep breath and hold it until you finish reading the next two paragraphs. While you were reading the initial part of this lab, your lower level lung control behavior was steadily commanding your chest muscles to expand to pull fresh air in, and then to contract to push your exhaust out. This behavior chugs along, nicely keeping you alive, without your brain thinking too much about it. Having this awesome lower level behavior, frees your brain up to think about more demanding things, like reading this paragraph.

Then bam! You fall out of a canoe and a higher level water-survival behavior (diving reflex), rapidly closes off your air intake to keep you from pulling dirty river water into your lungs. It subsumed control over your lung actuators. Subsumption based bio-control and just saved your life! The ability to have both behaviors (and in practice many behaviors) running at the same time, gives people and robots the ability to robustly survive and adapt to a dynamically changing world. You can stop holding your breath now, relax your subsumption override, and let the lower level behavior take over and breath for you while you read the rest of this lab.

Brooks proposed to run many robot-behaviors, from low to high, all in parallel, all able to see the same sensors and drive the same actuators:

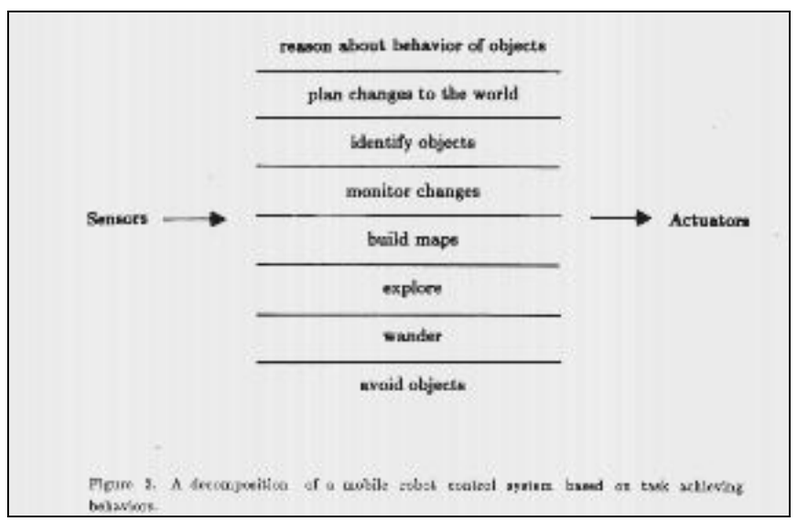


Figure 2. A decomposition of a mobile robot control system based on task achieving behaviors.

ENGR3390: Fundamentals of Robotics Tutorial (THINKLab - behaviors, arbitration, OCU) 2020a

This one bold step allowed robots, for the first time, to drive around and interact with a dynamically changing world, in real-time, using non-room size computers. Specifically, not just in a computer simulation sped up, but in actual real-time.

Within this form of control system, you will write a set of simple behaviors as functions and then weave them together into a structure like the one in Brook's original paper:

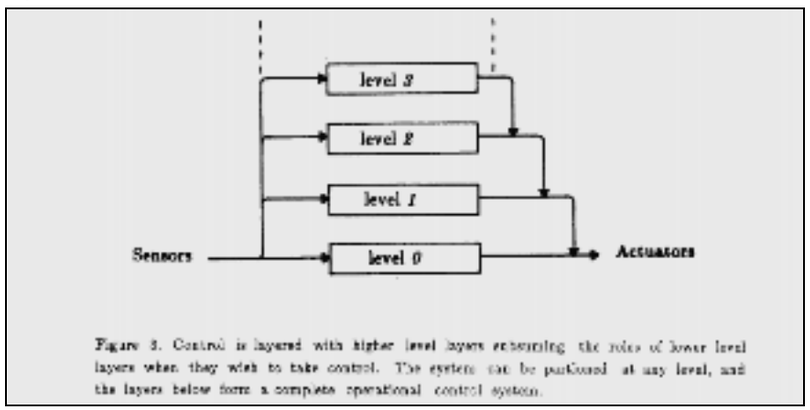


Figure 3. Control is layered with higher level layers subsuming the roles of lower level layers when they wish to take control. The system can be partitioned at any level, and the layers below form a complete operational control system.

We will use finite state machines (FSM) encoded into Arduino functions to hold each behavior:

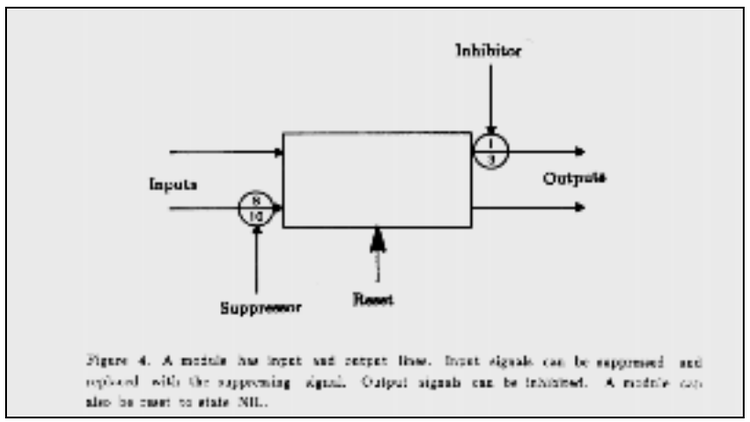


Figure 4. A module has input and output lines. Input signals can be suppressed and replaced with the suppressing signal. Output signals can be inhibited. A module can also be reset to state NIL.

Eventually ending up with a system of about the complexity shown below to drive your THINK lab-cart's robot tugboat.

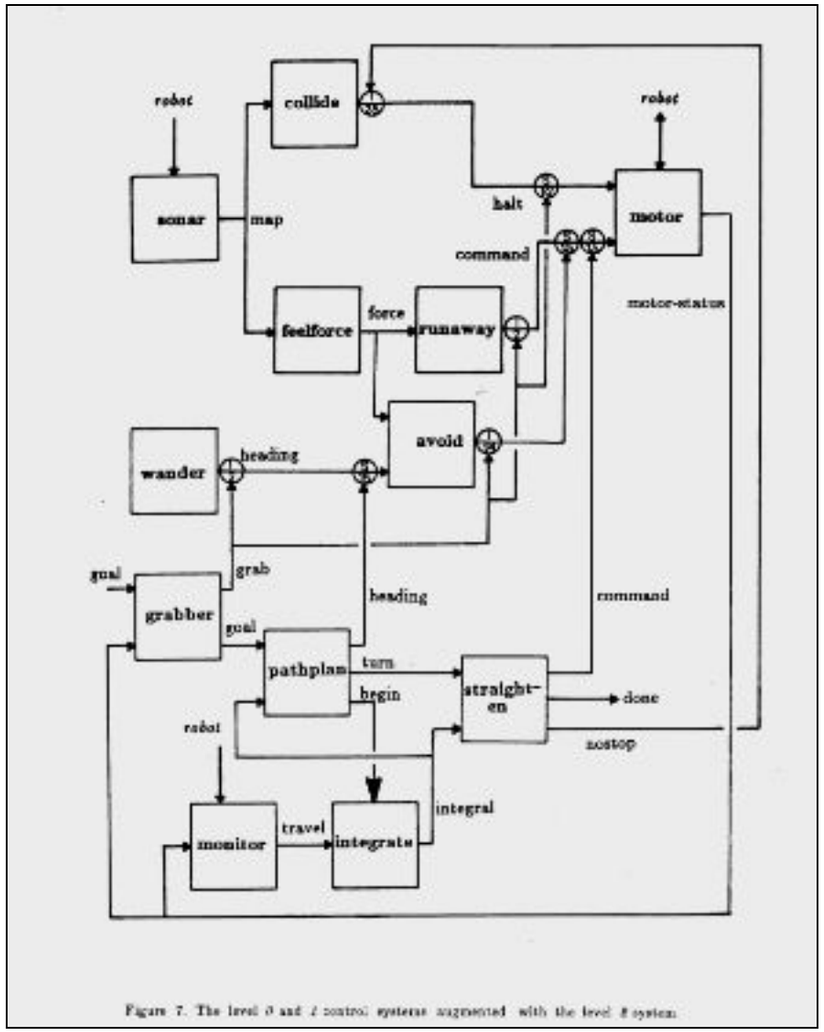


Figure 7. The level 0 and 1 control systems augmented with the level 2 system.

ENGR3390: Fundamentals of Robotics Tutorial (THINKLab - behaviors, arbitration, OCU) 2020a

In this lab, we will give you the SENSE and ACT parts of the tug code and work with your team to develop and give you hands-on experience with writing the THINK part, namely, the following set of simple robot-tugboat behaviors:

-Go to a fixed heading
-Avoid obstacles in front of boat
-Follow a target (a brightly colored purple NarWhal plush toy)

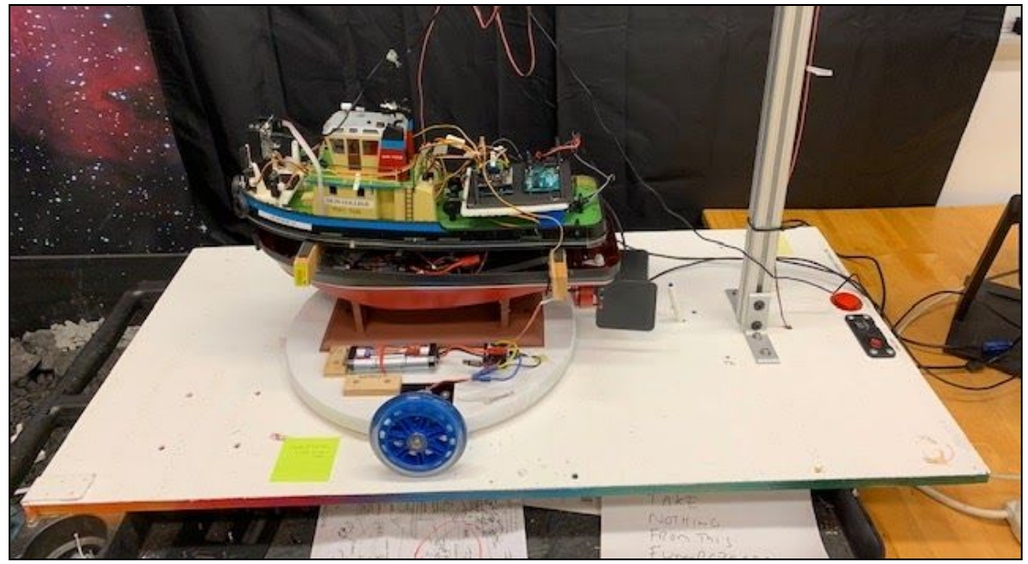
And help your team write two simple arbiters; one to control the commanded rudder angle of the tug and a seperate one to control the speed of its propellers.

As a stretch goal, your team can also write a simple radio link for a Laptop based OCU so that you may command your Robot Tug from a safe shoreline, without a long USB cable to the boat.

Finally, as in all 4 labs, we will ask you to, in parallel with your lab work (in part 2 of this assignment), too keep advancing your MATLAB programming skills by carrying out an additional set of simple MATLAB tutorials. The Instructors realize that this is not really helping in this particular lab, but will be very important for the other labs your team will do next and as such need your team to keep pace with the other labs teams in building your MATLAB skill sets.

Think Lab

This lab contains a fully functional robotic tugboat mounted on a driven turntable. The tugboat has a PixyCam and an array of Sharp Infrared range sensors as its main perception suite, a multiple arduino cognition/vehicle control bay and a set of embedded speed controls for the behavior lights, sound system, rudder and propellers. The multiple Arduino bay will allow your team to choose an optimal system configuration for your robot boat. You may use one Arduino each for SENSE-THINK -ACT, by having all three talk over an IC2 serial link, or you can run all of your code on a single Arduino in one large file, or you can try something in between:



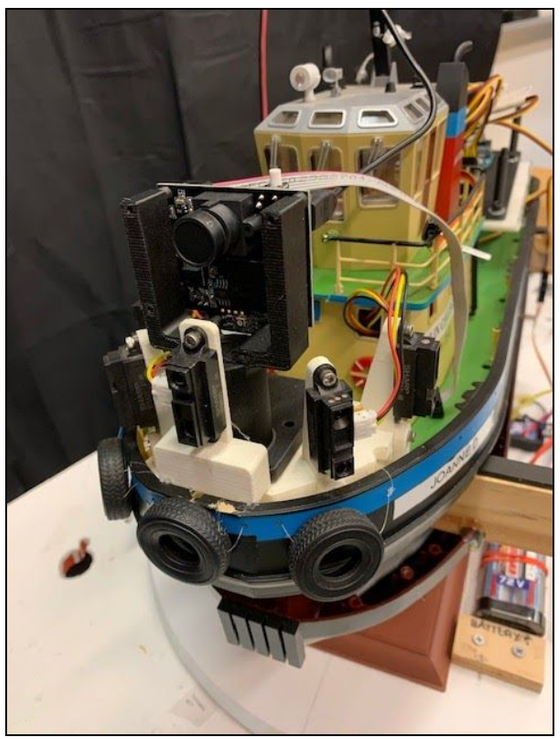
Think Lab Hardware

NarWhal Target

Description of Robot Tug SENSE code functions

The THINK lab robot tugboat has three types of sensors on it, an IR-Range suite to see and help avoid obstacles, an Advanced Camera to find the purple Narwhal and a turntable mounted Potentiometer to stand in for the boats compass to provide a boat heading. Each suite and where to find Arduino code for it is described below:

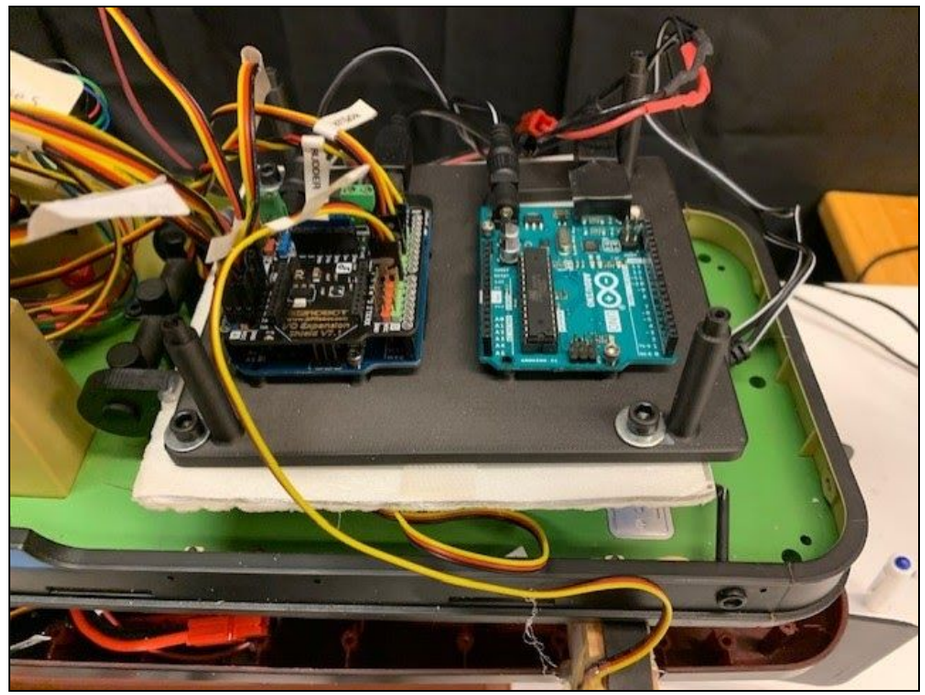
1. There is an array of SHARP Infra-red range sensors on the bow.



Sharp IR Range Sensor Array and PixyCam

They can be used for obstacle avoidance to give you a set of ranges to whatever is in front of your tug. You can find all of the information and Arduino code you need to connect to and use your Sharp IR sensors here: https://www.makerguides.com/sharp-gp2y0a21yk0f-ir-distance-sensor-arduino-tutorial/

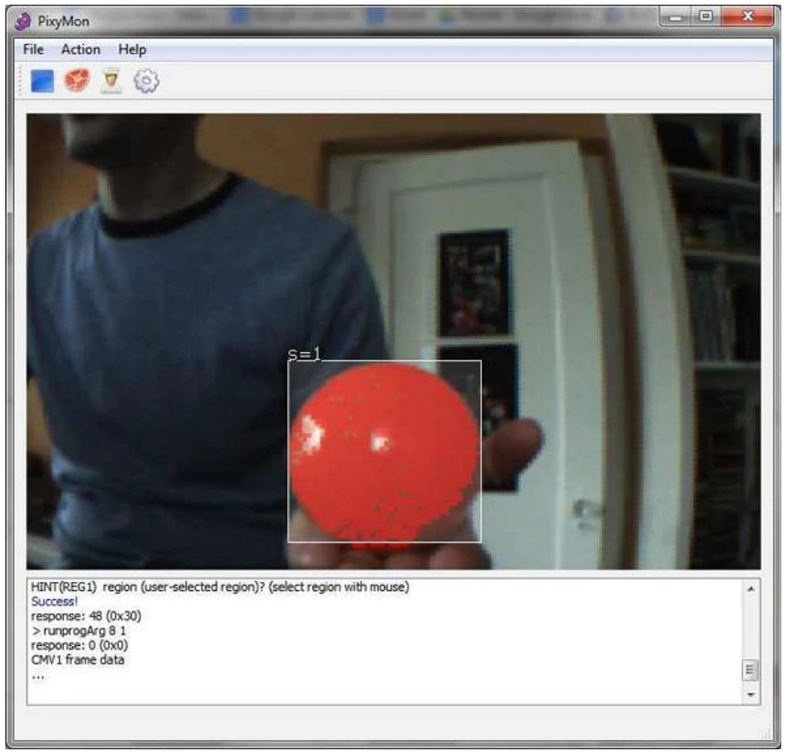
Wiring for the Sharp IR range suite to your Arduino IO Shield (analog Pins A1,A2,A3,A4,A5 as shown here:



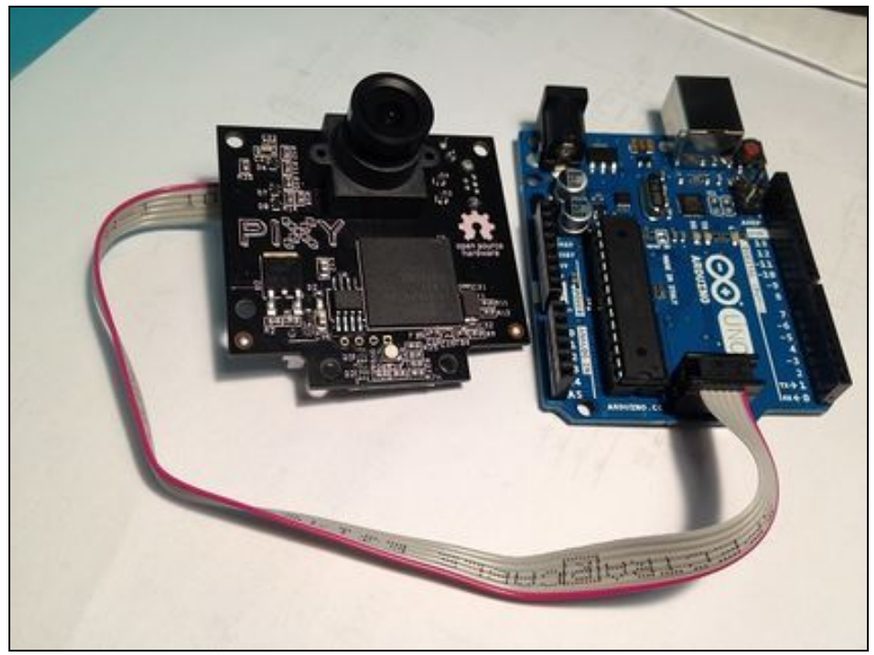
Wire all servo motors and sensors to attached IO shield

2. There is a PixyCam to let you find and track the purple NarWhal. The PixyCam is an embedded microprocessor based computer vision sensor that will let you find and track multiple colored blob targets from your Arduino. A very detailed tutorial of how to hook your PixyCam up to your Arduino and the code needed to talk with it can be found here: https://docs.pixycam.com/wiki/doku.php?id=wiki:v1:Hooking_up_Pixy_to_a_Microcontroller_-28like_an_arduino-29

   A Typical color tracking output of PixyCam shown here:



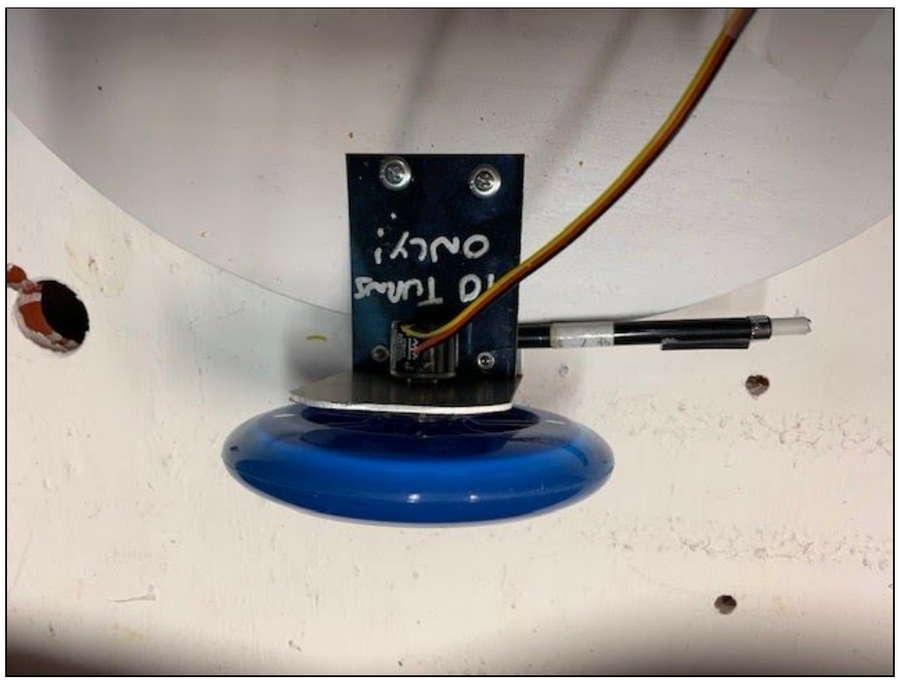
While you will wire your PixyCam to your Laptop for programming over a provided USB cable. Once you have it fully trained up then wiring your PixyCam to your Arduino is done with provided cable as is shown here:



In addition to the PIXYCAM website, you can also refer to the Fun-Robo lab 1 write up for a detailed description of how to train your PixCam to reliable track the purple whale and ignore all other objects.

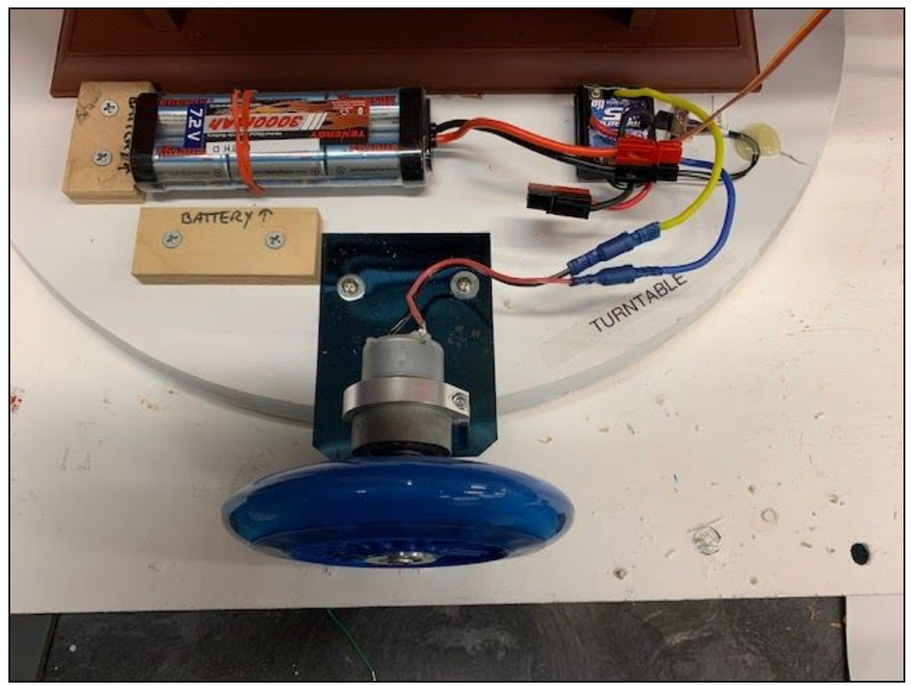
3. There is a multi-turn potentiometer to tell you the boats current heading with respect to the test cart. A detailed description with Arduino code of how to hook up the turntable heading potentiometer can be found here: https://www.arduino.cc/en/tutorial/potentiometer.

   The heading turntable Potentiometer shown here:



Wiring your heading Potentiometer to Analog input pin A0

4. There is a Marine Viper servo speed control to drive the turntable drive motor. It is basically an open loop DC motor speed control. It's pre-wired to the turntable drive wheel motor. The battery for this motor sits in a cradle near the hull on the turntable. There is a three wire servo cable coming up the outside of the boat's hull that you can plug into any PWM capable pin on the Arduino IO shield:



Turntable drive motor and MarineViper speed control

5. There is a standard RC servo motor inside the Tug connected to the twin rudders. Its three wire servo cable runs up through a hole on the deck and can be plugged into any PWM capable pin on your IO expansion shield

6. The twin propellers are driven by a second Marine-Viper speed control unit located inside the Tugs hull. Its propulsion battery sits inside the hull next to the speed control. There is a single three wire servo cable that runs up through a hole in the deck that can also plug into any PWM capable pin on your IO expansion shield.

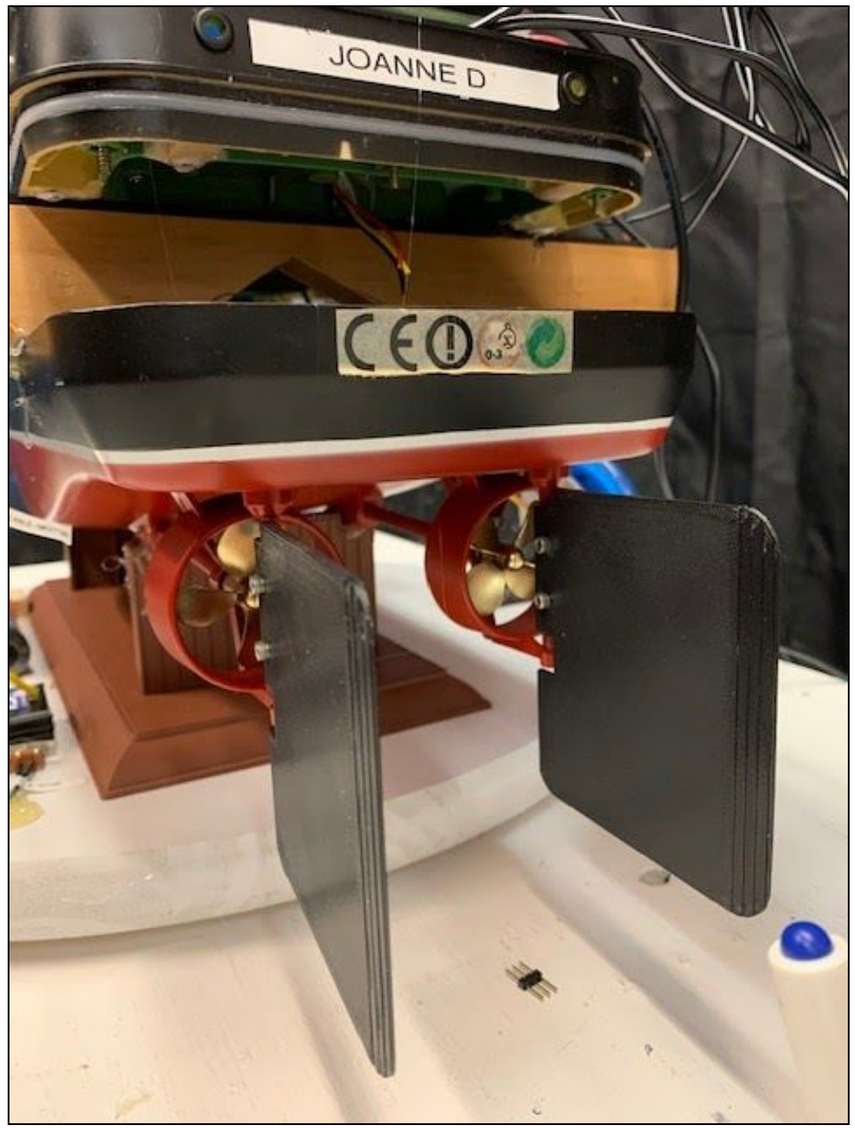
7. There is also a set of RGB super bright LEDs inside the wheel house of the tug. As a stretch goal, you can hook these up to your IO shield to and change the colors based on what behavior you tug is currently doing.

Description of Robot Tug ACT code functions

The robot tugboat test stand uses two dual channel MarineViper speed controllers to run both the tug's propellers and rudders and to also turn the tug's simulation turntable. Both Marine Viper motor speed controllers take two simple Arduino PWM servo signals as inputs. The internal Marine Viper will use one PWM input channel to control the dual propellers speed (reverse-stop-forward) and one to control the dual rudder's positions (hardport-center-hardstarboard).

The second Marine Viper will use a single channel to control the speed of the turntable heading motor (turnright-stop-turnleft).

More information on sending out a PWM signal from the Arduino's PWM pins can be found here:
https://www.instructables.com/id/Arduino-Servo-Motors/



Tug twin rudders and propellers

The main goal of a dry dynamic robot boat test setup such as this, is that it will lets you test your robot flight code with full hardware in the loop, without having to get wet or loose the boat in a lake.

In order to make this work, you will need to write a little bit of extra code that makes the turntable rotate to simulate the boat turning on the water.

You will need to write a bit of simulated motion code so that the turntable only turns if the rudders are not at zero and the propellers are on. The turntable should also turn one way if the propellers are moving the boat forward and the reverse way if it is backing up.

Given the collection of SENSE and ACT functions described above, develop a one, two , three or 4 Arduino robot control architecture for your tug (your team gets to choose). First draw out a simple block diagram as to what each Arduino is doing and then use it to create a detailed block diagram of what functions are running on each Arduino. Used your existing Simple Arduino Robot Controller as the base code from which to cast the code you will write for each Arduino in your system. Please show instructor and Ninjas block diagrams before beginning to write serious amounts of robot boat code. Please bear in mind that if you want to explore wireless communications, the THINK Arduino needs to carry an xBee wireless radio shield and will act as the communication link to your off-board laptop. You can use your Laptop as a simple OCU or you may push any heavy processing you might want to take on to it as well.

Once you have your general architecture blocked out, your lab group can divide into three subteams, one for SENSE, one for THINK and one for ACT.

The SENSE team needs to write two functions, one that gets obstacle ranges from the Sharp IR sensors and one that gets the NarWahl heading form the PixyCam.

The ACT team needs to write two functions, one that sets the rudder position and the propellor speed based on a desired boat heading and velocity, and another that rotates the turntable based on the rudder angle and the propellor direction.

The Think team will need to tie the SENSE and ACT functions together via some form of behavioral controller as will be described in the following section.

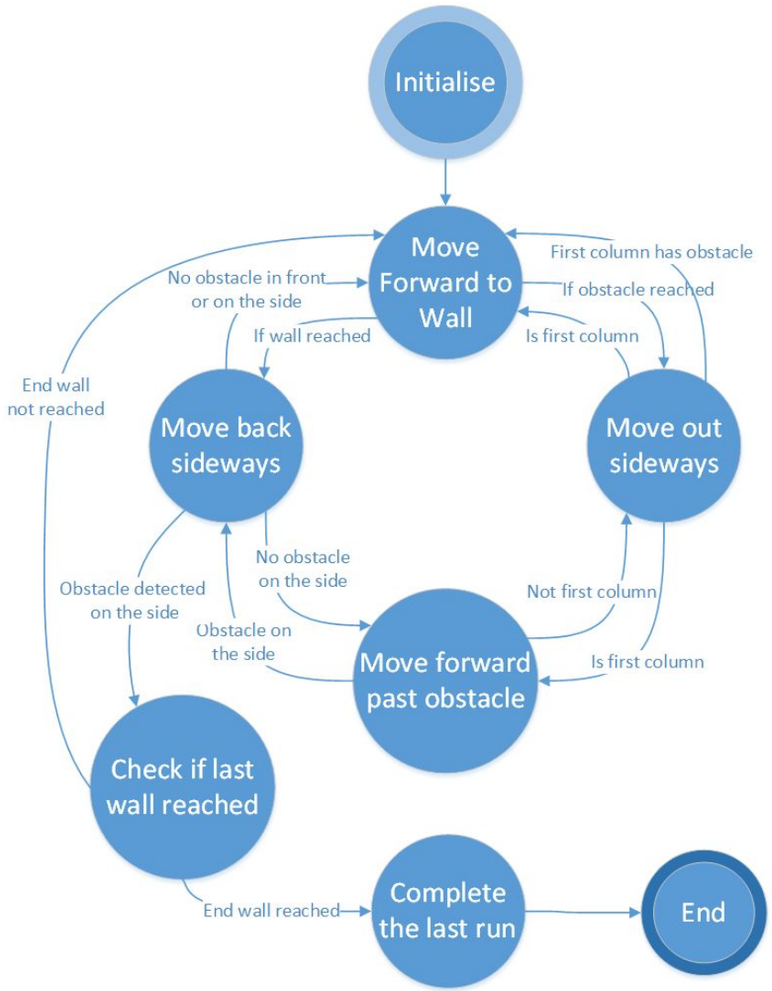
Fundamental robot behaviors

Complex robots usually use some form of behavioral control.

Using the concept that each behavior is a function; with the inputs to the function coming from data collected by your SENSE code and its outputs are the ACT variables, namely; desired rudder angle, desired propellor speed, turntable speed, (and optionally desired LED code (blinking lights) and desired sound from attached buzzer), write a minimum set of behavior functions that will express the following robot behaviors:

-Go to a fixed heading
-Hunt for target
-Avoid obstacles in front of tug (i.e. turn away from close objects)
-Follow a target (a brightly colored marine creature plush toy)
-Emergency Stop

As a stretch goal, you may write any additional behaviors you might like. Please create a clear behavioral finite state machine diagram and show it to the course instructor, before diving too deep into the writing of the final demo tug code. Please clearly define each state, from the initial starting one to the final stopping one and make sure you label your state transition triggers. A representative example is shown below:



Example Robot Behavior FSM Diagram

Designing a simple behavior arbiter

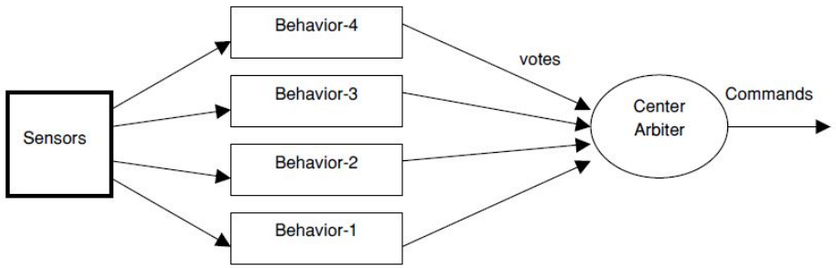
In a very practical sense, most actuators can only be in one position at any point in time. Interesting events happen when two or more behaviors (your functions from section above) try to command a single actuator to two separate desired positions at the same time. Some process has to successfully merge the two or more desired position requests and in your Arduino control code that function is called an **Arbiter**. A complex robot might have one for every major actuator. Your boat should have 2, one for rudder, one for propellor. Consider the case where a **GoTo Heading** behavior commands the rudder to 30 degrees to the right (a Starboard turn for you landlubbers) **and** the **Obstacle Avoid** behavior commands it to go 30 degrees to the left (a port turn). Do you average the command? Give one presidence? Blend the two with a probabilistic process? Turn rudder to a random angle and hope? Hit object dead-on? Ouch.

How to design a good arbiter for each actuator, is a very robot hardware and domain specific challenge and is an ongoing area of research.

Please read the paper; "Using Voting Technique in Mobile Robot Behavior Coordination For Goal-directed Navigation" Amin, Tar, Mamat for a good overview (in course folder)

http://eprints.utm.my/id/eprint/1396/1/JT36D5.pdf

and then look up some other possible arbiter function options on-line, by searching **robot behavior arbiter:**



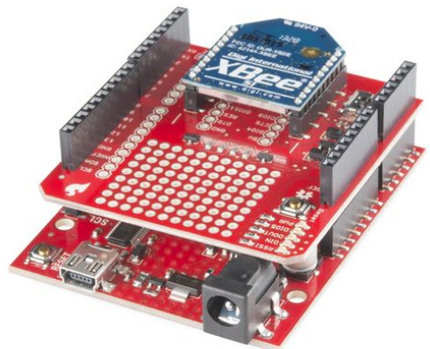
A Representative Behavior Arbiter

When you have a good grasp of what arbitration system you would like for each of your 2 Robo Tugboat actuators; propellor-speed, rudder-angle, (optional behavior-light pattern and behavior sound), please draw up a simple diagram of each and show to course instructor of ninja before diving into the heavy coding of them.

Developing a radio-linked laptop OCU

As a stretch goal. In order to be able to command and communicate with your Robot Tugboat from the safety of shore, you will need to design and code a; 1. radio linked Operator Control Unit (OCU) consisting of two parts; a laptop based OCU program (basically the Arduino Serial Monitor with lots of print statements) to allow the operator to turn on, off and initiate behaviors over a radio link and a 2. onboard tug THINK Arduino set of functions to allow the boats controller to receive your commands from the Laptop and send simple tugboat state data back to it.

In some very direct sense, you are creating a simple OCU for your Robot Tugboat. We can use a 900mhz xBee radio to allow serial communication between the Arduino on the tug and your Laptop:



xBee Radio https://www.sparkfun.com/products/12847

Hookup guide here;
https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide?_ga=2.169397857.1765882478.1537759530-744617176.1537759530

Prototype Arduino code here:

```
/*************************************************************
XBee_Serial_Passthrough.ino

Set up a software serial port to pass data between an XBee Shield
and the serial monitor.

Hardware Hookup:
  The XBee Shield makes all of the connections you'll need
  between Arduino and XBee. If you have the shield make
  sure the SWITCH IS IN THE "DLINE" POSITION. That will connect
  the XBee's DOUT and DIN pins to Arduino pins 2 and 3.

*************************************************************/
```

```
// We'll use SoftwareSerial to communicate with the XBee:
#include <SoftwareSerial.h>

//For Atmega328P's
// XBee's DOUT (TX) is connected to pin 2 (Arduino's Software RX)
// XBee's DIN (RX) is connected to pin 3 (Arduino's Software TX)
SoftwareSerial XBee(2, 3); // RX, TX

//For Atmega2560, ATmega32U4, etc.
// XBee's DOUT (TX) is connected to pin 10 (Arduino's Software RX)
// XBee's DIN (RX) is connected to pin 11 (Arduino's Software TX)
//SoftwareSerial XBee(10, 11); // RX, TX

void setup()
{
  // Set up both ports at 9600 baud. This value is most important
  // for the XBee. Make sure the baud rate matches the config
  // setting of your XBee.
  XBee.begin(9600);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available())
  { // If data comes in from serial monitor, send it out to XBee
    XBee.write(Serial.read());
  }
  if (XBee.available())
  { // If data comes in from XBee, send it out to serial monitor
    Serial.write(XBee.read());
  }
}
```

Initialization

```
#include "SerialPort.h"
#include <iostream>

char* portName = "\\\\.\\COM20";

//Declare a global object
SerialPort *arduino;

int main(void)
{
  arduino = new SerialPort(portName);
  std::cout << "is connected: " << arduino->isConnected() << std::endl;
}
```

Sending Data

```
#define DATA_LENGTH 255

#include "SerialPort.h"
#include <iostream>

char* portName = "\\\\.\\COM20";

//Declare a global object
SerialPort *arduino;

//Here '\n' is a delimiter
char *sendString = "Hello World\n";

int main(void)
{
  arduino = new SerialPort(portName);
  if (arduino->isConnected()){
    bool hasWritten = arduino->writeSerialPort(sendString, DATA_LENGTH);
    if (hasWritten) std::cout << "Data Written Successfully" << std::endl;
    else std::cout << "Data was not written" << std::endl;
  }
}
```

Receiving Data

```
#define DATA_LENGTH 255

#include "SerialPort.h"
#include <iostream>

char* portName = "\\\\.\\COM20";

//Declare a global object
SerialPort *arduino;

//Here '\n' is a delimiter
char receivedString[DATA_LENGTH];

int main(void)
{
  arduino = new SerialPort(portName);
  if (arduino->isConnected()){
    int hasRead = arduino->readSerialPort(receivedString, DATA_LENGTH);
    if (hasRead) printf("%s", receivedString);
    else printf("Error occured reading data");
  }
}
```

As always, please see an instructor or Ninjas for help with any part of the above.

Final demo: The final demo has two parts. In part 1, please demo each simple behavior operating independently, namely:

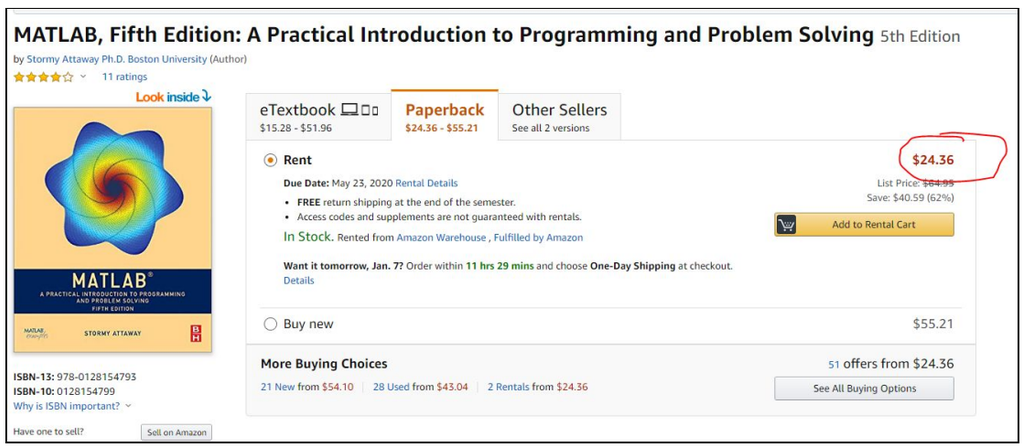
-Go to a fixed heading
-Avoid obstacles in front of tug (i.e. turn away from close objects)
-Follow a target (a brightly colored marine creature plush toy)
-Emergency Stop

In part 2, the instructor will move purple NarWhal in front of your boat while Ninja moves floating iceberg obstacles in front of it as well. For this demo you should have all 4 tug behaviors running concurrently and have your arbiter passing rudder and speed control to whichever of them will allow the tug to continue to pursue the NarWhal without hitting any icebergs. Your boat should turn, backup and turn, go fast when open and slow when blocked and relentlessly and head toward NarWhal whenever it can be seen. If it can't be seen, your boat should turn widely to port and then widely to starboard hunting for it.

MATLAB Skill Building Tutorials

While you are working on your lab hardware, we would like you to continue developing your MATLAB skills in anticipation of needing them during the 4 hardware labs and the final project. The On-Ramp was a nice introduction, but you can only learn depth in core skills technical skills by use and repetition. You will get this depth in MATLAB for this course by working through what is probably the best self-paced learning reference book Matlab:

https://www.amazon.com/MATLAB-Practical-Introduction-Programming-Problem/dp/0128154799/ref=sr_1_5?crid=27M7QRHEGC9MD&keywords=matlab&qid=1578329131&s=books&sprefix=Matlab%2Caps%2C150&sr=1-5



When working on a single laptop screen its quite helpful to have the physical book to prop open next to it while working. You can buy a cheap used one from Amazon at the link above. But if you prefer digital and have a second monitor (or don't mind a lot of screen toggling) a 4th edition.pdf of this book is available in the course folder.

Working in 4 two week long segments (while doing the 4, 2 week long hardware labs) we will ask you to read through the following chapters and do the short tutorial examples in them in the following order. Regardless of which lab you start with, it would be best for you to proceed through the chapter tutorials in the order shown. If you already are pretty proficient in MATLAB via other Olin courses, you can just quickly scan this material for a fast refresher. There are no set deliverables here, we are just trying to get each robot lab team up to a reasonable common level of MATLAB coding skill at a reasonable pace. If you are a novice MATLAB programmer, it is recommended that you work through the material fairly consistently, but feel free to skip over any parts you don't feel are relevant, you can always return to them later. Again there is no formal deliverable for this work, but you will both learn a lot more and get a lot more out of this course if you aren't constantly asking your fellow teammates or the Ninjas how to **Plot** or how to write a simple **Switch** structure.
Please see course instructor or Ninjas for MATLAB help as needed.

Lab Week 1-2: (Ch 1, 2 and 3)

Chapter 1: *Introduction to MATLAB begins by covering the MATLAB Desktop Environment. Variables, assignment statements, and types are introduced. Mathematical and relational expressions and the operators used in them are covered, as are characters, random numbers, and the use of built-in functions and the Help browser.*
Chapter 2: *Vectors and Matrices introduces creating and manipulating vectors and matrices. Array operations and matrix operations (such as matrix multiplication) are explained. The use of vectors and matrices as function arguments, and functions that are written specifically for vectors and matrices are covered. Logical vectors and other concepts useful in vectorizing code are emphasized in this chapter.*
Chapter 3: *Introduction to MATLAB Programming introduces the idea of algorithms and scripts. This includes simple input and output, and commenting. Scripts are then used to create and customize simple plots, and to do file input and output. Finally, the concept of a user-defined function is introduced*

with only the type of function that calculates and returns a single value.

Lab Week 3-4: (Ch 4 and 5)

Chapter 4: *Selection Statements introduces the use of logical expressions in !f statements, with else and elseif clauses. The switch statement is also demon strated, as is the concept of choosing from a menu. Also, functions that return logical true or false are covered.*
Chapter 5: *Loop Statements and Vectorizing Code introduces the concepts of counted (for) and conditional (while) loops. Many common uses such as summing and counting are covered. Nested loops are also introduced. Some more sophisticated uses of loops such as error-checking and combining loops and selection statements are also covered. Finally, vectorizing code using built-in functions and operators on vectors and matrices instead of looping through them, is demonstrated. Tips for writing efficient code are emphasized, and tools for analyzing code are introduced.*

Lab Week 5-6: (Ch 6 and 7)

Chapter 6: *MATLAB Programs covers more on scripts and user-defined functions. User-defined functions that return more than one value and also that do not return anything are introduced. The concept of a program in MATLAB which consists of a script that calls user-defined functions is demonstrated with examples. A longer menu-driven program is shown as a reference, but could be omitted. Subfunctions and scope of variables are also introduced, as are some debugging techniques. The Live Editor is introduced.*
Chapter 7: *String Manipulation covers many built-in string manipulation functions as well as converting between string and number types. Several examples include using custom strings in plot labels and input prompts.*

Lab Week 7-8: Wrap up (Ch 9 an 10)

Chapter 9: *Advanced File Input and Output covers lower-level file input/output statements that require opening and closing the file. Functions that can read the entire file at once as well as those that require reading one line at a time are introduced and examples that demonstrate the differences in their uses are shown. Additionally, techniques for reading from and writing to spreadsheet files and also .mat files that store MATLAB variables are introduced. Cell arrays and string functions are used extensively in this chapter.*
Chapter 10: *Advanced Functions covers more advanced features of and types of functions, such as anonymous functions, nested functions, and recursive functions. Function handles, and their use both with anonymous functions and function functions are introduced. The concept of having a variable number of input and/or output arguments to a function is introduced; this is implemented using cell arrays. String functions are also used in several examples in this chapter. The section on recursive functions is at the end and may be omitted.*

Wrap up MATLAB tutorials. Upon completing all of these basic skill building tutorials, you will have acquired a pretty solid undergraduate MATLAB coding skill set and will be well on your way to creating your own elegant robot code. You can continue your self-learning of more in-depth MATLAB skills by reading through and trying some of the more sophisticated features of MATLAB presented in the other chapters of the book.