Programming in C – Assignment 2
Marco Programming Assignment

By Jon Ambrose (07503425) & Andrew Fester (0750) (JAAF Software)

Part 1 - MARCO

Contents

PART 1

Marco

Our approach to the Marco Programming Assignment

Our approach will be based on a development, of a piece of software, which has many functions in it to make it a complete piece of software.

Most of these functions will be a core part of the objective of the main program; these include: the analogue joystick control, bumpers, brightest light following, and white line following. Some of these functions belong to each other in terms of the white line following, and when the robot hits an obstacle (bumpers function) it will stop. More detail of these will be stated in the finite state diagrams in our design section.

We will develop each sub task in an individual file and make it so it works as a function. Therefore, when we consolidate the menu in the final file, the code to copy into each case statement, will be much simpler.

By developing the sub tasks in each of their own file, it will help to debug and stop the confusing issues in terms of what other tasks are going to be running when the c file is complied and ran through the marco. It also stops you getting fairly inaccurate results. It also helps because say you have the joystick control in one file and coded each function in, when it comes to run the complied C program you would need to calibrate the joystick each time, which could be a right pain.

By using the separate file for each function development technique it means that version control will a lot easier to handle, because you only focus on one part of the program.

Another advantage over using separate files to code each function means you can easily test data – i.e. read the values of the light source, whether on white line or not or joystick control etc.

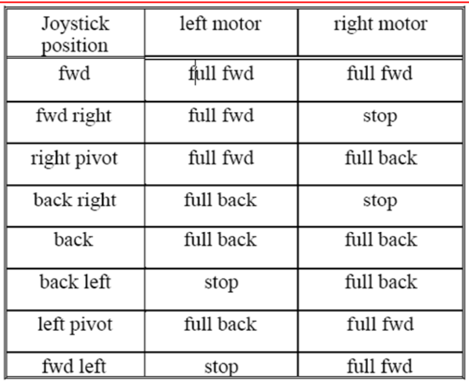
Once we have developed each of the main parts of the program, these will be consolidated into one big master file – which will be integrated via a menu in lcurses. The menu will be constructed with a case statement. Each task will be in a separate statement, which will be linked to “Actions”. More will be explained in the finite state diagram.

Our design to the Marco Programming Assignment

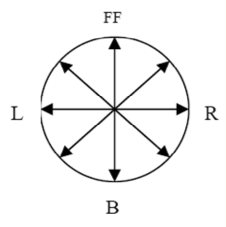
Joystick module

- Read the Analogue joystick controller and look up in an array as to what values to write to the motors.

```
int array [6][3] =
  {
    {4073 , 4073 , 2048}, /*{r-lfw, r-fw    , r-rfw},*/
    {4073 , 2048 ,   20}, /*{r-l  , r-center, r-rb },*/
    {  20 ,   20 , 2048}, /*{r-lbw, r-bw    , r-rbw},*/
    {2048 , 4073 , 4073}, /*{l-lfw, l-fw    , l-rfw},*/
    {  20 , 2048 , 4073}, /*{l-l  , l-center, l-r  },*/
    {2048 ,   20 ,   20}, /*{l-lbw, l-bw    , l-rbw},*/
  };
```



Joystick position	left motor	right motor
fwd	full fwd	full fwd
fwd right	full fwd	stop
right pivot	full fwd	full back
back right	full back	stop
back	full back	full back
back left	stop	full back
left pivot	full back	full fwd
fwd left	stop	full fwd



Bumpers

- Read the digital marco rack that is connected to the robot. The robots bumpers are read by the digital device in D0 (Left Bumper) and D1 (Right Bumper).
- If either D0 – 0x01 = = 1 (AND OR) D1 – 0x02 == 2, then stop values which are “2048” are written to the right and left motors, to make the robot stop.

White Line Follow

- Read the digital inputs D2 – 0x04 (left IR) andD3 – 0x08 (right IR) which is connected by IRSW 1 (left) and 2 (right).
- These values are then integrated into a case statement to make the robot motor values set to what IR has been triggered to make 0x04 to make it turn left and 0x08 to make it turn right.

Brightest light following

- Read the digital marco rack that measures the stepper motor when it hits far left or far right clicks. This is called the "Stepper Motor Switch – left and right".
- Stepper motor switch = D4 – 0x10 (left) 16 decimal.
- Stepper motor switch = D5 – 0x20 (right) 32 decimal.
- I then need to make the stepper motor move to one side (either left or right) – I have chosen it to move all the way to the right. After this has completed I
- Will put a counter in the sequence so that it measures how many counts it makes to get to left stepper motor switch, this increments every time it goes round the loop.
- When it's reached the left stepper motor it will go into another while loop which will then count how many cycles it will take to get back until it reaches the right stepper motor.
- I will then compare the two counts and if they are not equal then display error and exit.
- The program will then enter another while loop and this is what will repeat it going left to right, right to left all the time.
- The next bit of code will need to determine what direction to travel in of the sweep to go to the left switch; therefore I will be able to measure the direction of the robot it has to go by using the counter theory. I will find out how many cycles it takes to get one side. The direction will be calculated by doing the cycles divided by 4.
- Therefore if the current eye data (light meter) is greater than a typical high brightness, then it will enter an if statement which will say if quad (counter) is greater than 0 but less than 2.75 – it will go far left. And between 2.75 and 8.25 it will go straight on. If greater than that it will go right.
- The above code will also be copied for the sweep to the right switch, but instead of incrementing the counter it decrements. By doing this we are making sure the same area is always the same motor values. Either left, forward, right.

Testing

Checking the program works with the assignment.

Conclusions/Review

We have learnt that hardware is very hard to make right all the time – different marco robots behave differently. We have learnt that we could make our source code alot smaller, but because we were testing the robot alot and it worked, we decided to key the source code longer.

Appendix

A plan of the Macro Programming Assignment – See attached

Timesheet

date	Start Time	Finish Time	Who present	Tasks done
19/04/2008	12.00	14.00	Jon Ambrose & Andrew Fester	
26/02/2008	12.00	14.00	Jon Ambrose & Andrew Fester	
04/03/2008	12.00	14.00	Jon Ambrose & Andrew Fester	Controller
11/03/2008	11.00	17.00	Jon Ambrose & Andrew Fester	moving the Marco
24/03/2008	11.00	17.00	Jon Ambrose	Bumpers
25/03/2008	11.00	17.00	Jon Ambrose	
26/03/2008	11.00	17.00	Jon Ambrose	Brightness light follow
27/03/2008	11.00	17.00	Jon Ambrose	
28/03/2008	11.00	17.00	Jon Ambrose	White Line Following
03/04/2008	12.00	18.00	Andrew Fester	path remembering
12/04/2008	13.00	18.00	Jon Ambrose & Andrew Fester	general program integration
13/04/2008	16.00	19.00	Andrew Fester	
08/04/2008	12.00	14.00	Jon Ambrose & Andrew Fester	
15/04/2008	12.00	14.00	Jon Ambrose & Andrew Fester	Clear bugs and RTM test
16/04/2008	10.00	21.00	Jon Ambrose & Andrew Fester	
16/04/2008	21.00	24.00	Andrew Fester	
17/04/2008	0.00	7.00	Andrew Fester	
17/04/2008	12.00	-	Jon Ambrose	Commenting and report
18/04/08	-	8.00		

PART 2

Research

Rodney Brooks:

Rodney brooks work focused mainly on biologically inspired robotic architectures which at the time was his argument against symbolic approaches to intelligent machines. He is responsible for the development of Subsumption architectures and augmented finite state machines as part of the development of his intelligent machines.

His work in robotics, first published in 1986 and subsequently elaborated upon in a series of highly influential papers, inaugurated a fundamental shift in artificial intelligence research. Brooks has argued strongly against symbolic processing approaches to creating intelligent machines, which had been the focus of AI since the days of Alan Turing, directly tracing back to the work of Gottlob Frege. Instead, Brooks has focused on biologically-inspired robotic architectures (e.g., the Subsumption architecture) that address basic perceptual and sensorimotor tasks. These had been largely dismissed as uninteresting by the mainstream AI community, which was far more interested in reasoning about the real world than in interacting with it. Conversely, Brooks argued that interacting with the physical world is far more difficult than symbolically reasoning about it.

Subsumption architectures:

This is simulating intelligent behaviour by breaking it down into lots simple behaviour modules which are then put into layers. The higher layers of this system would work toward goals and the further down you go the more it works like reflexes. For example I can decide to walk somewhere but there is a complex system of muscle usage in order for me to achieve that goal.

The main argument behind the subsumption architecture is that intelligent machines would have to be flexible. Building this intelligence from lots of small modules will, when combined allow for a wider range of more complex tasks to be accomplished.

For example, a robot's lowest layer could be "avoid an object", on top of it would be the layer "wander around", which in turn lies under "explore the world". Each of these horizontal layers access all of the sensor data and generate actions for the actuators — the main caveat is that separate tasks can suppress (or overrule) inputs or inhibit outputs. This way, the lowest layers can work like fast-adapting mechanisms (e.g. reflexes), while the higher layers work to achieve the overall goal. Feedback is given mainly through the environment.

2. The Behaviour Based Approach

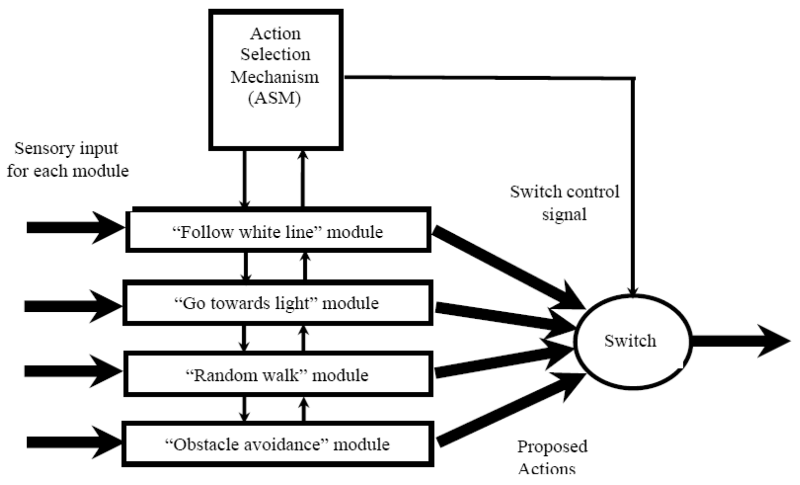

Action Selection Mechanism (ASM)
Sensory input for each module
Switch control signal
"Follow white line" module
"Go towards light" module
"Random walk" module
"Obstacle avoidance" module
Switch
Proposed Actions

Figure 2: The Behaviour Based Approach

Attributes of the architecture

The main advantages of the methodology are:

- the modularity, (an engineering technique that builds larger systems by combining smaller subsystems)
- the emphasis on iterative development & testing of real-time systems in their target domain, and
- the emphasis on connecting limited, task-specific perception directly to the expressed actions that require it.

These innovations allowed the development of the first robots capable of animal-like speeds.

Main disadvantages of this model are:

- the inability to have many layers, since the goals begin interfering with each other,
- the difficulty of designing action selection through highly distributed system of inhibition and suppression, and
- the consequent rather low flexibility at runtime.

Augmented finite state machines:

This is where the simple behaviours are hierarchically organised to allow for more complex behaviours to develop. Timers are added to these basic Finite State Machine (FSM) to induce salient, coherent behaviour. Inhibition and Suppression between behaviour modules provides distributed control, and therefore this system results in Pre-wired patterns of behaviour.

http://en.wikipedia.org/wiki/Subsumption_architecture

http://en.wikipedia.org/wiki/Rodney_Brooks#Publications

http://ai.eecs.umich.edu/cogarch3/Brooks/Brooks_AFSM.html

http://ai.eecs.umich.edu/cogarch0/subsump/method.html