**The Robot: Technical Report**

DATE: 29th APRIL, 2015

COURSE: APSC 1299

Created by: Umair Rashid

Xiaoyou Liang

Ridhi Dalla

The Robot Competition is a project that requires students who are taking Micro-controllers course, to work in groups of three and program a robot using the C language. Students must use MPLAB to write C code that allows the robot to make its way through a black track that includes specific obstacles. The robot provided uses five ADC (Analog to Digital Converter) sensors that are horizontally aligned in front of the robot, to sense the track. The ADC sensor uses a threshold (a digital number) that is set by the user to detect the colour he/she intends for it to sense. The robot also consists of five LED lights, one corresponding to each sensor, that light up when the related sensor senses the track. The track consists of numerous challenges and obstacles that the robot should be able to smoothly complete. Our task for this project was to come up with a unique strategy to improve and alter the original code in motor\_control.c that uses functions from sumovore.c, provided to us with the robot, such that the robot is able to complete the track without having the need to be guided by a person or getting confused between different challenges.

We initially approached this this task by incorporating the use of timers in our code to be able to get through each task. We were able to use this approach to get each obstacle to work out individually; however, when we tried to get the robot to run through multiple challenges at the same time, it did not do too well as it was getting confused between challenges. To fix this problem, we tried a second approach that required minimal use of timers to be able to get the robot to run through the track, it worked out fine. Following are our different approaches and solutions that we came up with in order to finish the competition.

**First Approach**

Initially, we programmed the robot to overcome each challenge using timers.

**90⁰ turns**: if the robot senses the track from the first two (0b11000u), or the first three (0b11100u), or the first four (0b11110u) LEDs, it enters a function called turn\_90\_left(). In this function, a timer is opened and is run for 0.04s during which time, it is ordered to go straight forward. When the timer finishes, the robot checks sensors again. If the sensors sense all white during this time period, another timer is open for 0.36s and during this time period, the robot follows the following set of instructions. set\_motor\_speed(left, rev\_slow, 0); set\_motor\_speed(right, fast, 0);. These cause our robot to turn left for the time that the timer is run. After this, the program breaks out of this function. Similarly for when the sensor senses the track from the last two (0b00011u), last three(0b00111), or the last four(0b01111) LEDs, it enter a function called turn\_90\_right(). This function contains the same instructions as the turn\_90\_left() function. A timer is opened for 0.04s and it goes straight forward for that amount of time .Once again, when the LEDs sense all white, another timer is opened for 0.36s and the robot follow this set of instructions: set\_motor\_speed(left, fast, 0); set\_motor\_speed(right, rev\_slow, 0);. This makes the robot turn right.

Gaps and dead end: initially, a variable called has\_gap is set to 0. If the sensors sense no track (0b00000u), it enters a function called whether\_gap(). In this function the program checks the value of has\_gap. If the value is 0, a timer is opened for 0.8 seconds. During this time, it is set to go forward while constantly checking the sensors. If the sensors sense any initial condition of 90⁰ turns, it enters the turn\_90\_right() or turn\_90\_left()function. Between 0.48s and 0.8s, if it senses the track from any one of its sensors, it calls follow\_simple\_curves() which simply makes the robot follow the original track. And after 0.8s, if there is still no track, the value of has\_gap is set to 1. If the initial value of has\_gap checked by the function is 1, it calls turn\_90\_right() function twice making the robot turn around at the dead end and sets the value of has\_gap to 0.

Acute angle turns: If the robot senses the track from wither either third and fifth (0b00101u) or second, fourth and fifth (0b01011u), it calls a function turn\_30\_right(). In this function a timer is opened for 32ms. While this timer runs, the robot is set to go straight forward. After this 32ms, the robot checks the sensors again. If either the last three (0b00111u) or the middle three(0b01110u) sensors detect something, another timer is opened for 0.4s and the robot is set to go straight forward again, then check sensors. If all the sensors detect nothing (0b00000u), another timer is opened for 0.8s during which the robot follows the following commands: set\_motor\_speed(right,rev\_slow,0); set\_motor\_speed(left,fast,0);. This causes the robot to turn right for the set time period. This makes the robot to do the acute angle turn. This function contains the same instructions as the turn\_30\_left() function. The differences are the initial condition and turning left at the end. The initial condition is either first and third(0b10100u) or first second and fourth (0b11010u).

This code using timers was very complicated and hard to understand and edit. Every time a single change was made in the code, rests of the functions were also affected.

**Final Approach:**

**90 Degree and acute angle functions**

Since our acute angle function was not working very well. We decided to take a new approach. We started working on the acute angle and 90 degree functions together. Firstly, we focused on the 90 degree function (turn\_90\_right() and turn\_90\_left() ), and this time it was not based on timers. The way we changed it was, as soon as either the first two, three or four leftmost sensors detect the black track ,as shown in line 51 to 53, the program that runs the robot, calls the turn\_90\_left () (Line 232) function which asks the robot to go straight until the sensors detect all white and then turn left until they detect only the centre led. Therefore, the robot only turns until it is straight, and then it breaks out of the function. Instead of relying on timers for our robot, we opted for loops. We used do-while loops such as in the code for turn\_90\_left() and turn\_90\_right() for both going forward and turning as seen in line 232 and 183. This made a huge difference in our robot’s performance because now we did not have to worry about the exact time it takes for each step to execute, or whether the speed of the robot is slow or fast. We did the exact same thing for our turn\_90\_right () function as seen on line 183, and our 90 degree functions started to work very well.

Next we focused our attention on the acute angles. Initially, we focused on the conditions that would activate our acute angle turn function. We used a slow motion camera to find what conditions occur as robot approaches an acute angle. We tried placing our robot at different angles in order to find all the possible conditions. The conditions we found are in line 59 and 60 for left and line 66 and 67 for right turns. We realized that these conditions were triggered almost every single time the robot came close to an acute angle but there were times that it activated the 90 degree functions, and even if it dis activate the 90 degree functions, the robot would behave the same way because our acute angle functions were based on the same concept as the 90 degree function. These conditions called the acute angle function (turn\_30\_right() or turn\_30\_left()) in line 342 or 385, then the robot goes straight until it senses all white and then turns until only the middle sensor senses the track and then breaks out of the function. The reason we opted for a while loop in these functions is because of the unpredictability of the angles. If we had used timers we will have to adjust the time the robot should turn according to every possible angle, which is quite impractical. After the change, no matter what the angle is, the robot will turn appropriately.

**Gaps and dead end**

Our next target was to make sure the robot covered gaps in the track that were unto 8cm long. Our gaps were working well with the timers but the results were not consistent; it was having a hard time turning right away after a gap. Again, we used the while loops, and our initial code shrank from being thirty lines long to about fifteen lines long. It was a lot easier to read, write and understand. The way whether\_gap () function, line 277, works is that as soon as the robot detects all white, it goes straight, but if it senses anything else except white then it breaks out of the function. Again, we used a while loop starting from line 285 to 304. We also had to ensure that the robot would return if it does not find anything except white for a certain distance(8cm) by making a right turn. We used a timer to guide our turn. The way timers work here is, they are opened in the while loop for a certain amount of time. Once that time period comes to an end, it makes robot turn and then go straight until it senses anything as shown in line 294. We tried our gaps and they were very consistent with our trials. We focused a lot on 90 degree turns after the gap and it worked quite well.

**Deviation and tight curves**

As we were working on the gaps we realized that our robot was deviating towards the right. It seemed as if the left wheel was faster than the right wheel. Even though before starting this project, we tested our robot for deviation and we saw it deviating towards the right but it was over a very long distance and a very small deviation. Either way, we decided to fix this deviation since it was causing robot to move to the right as it was going straight when the gap function was activated. We modified the straight\_fwd() function on line 97. We tried different combinations to slow down the left wheel since it was faster than the right wheel and we finally found the right number to do so and using that, changed the left wheel’s speed modifier to slow it down a bit so that it can keep up with the right wheel. This change can be seen in line 99.

After fixing the deviation, we turned our attention to the tight curve. We used slow motion camera to check what happens as the robot crosses a curve. We realized that as our robot is goes straight when the path starts to curve causing the center led and the right-center or left-center led to light up as well depending on which direction the curve is turning in. As soon as this happens, our function turns the robot towards the left or right until only the center sensor senses the track. We can clearly see these conditions in lines 45 to 50. Once this condition occurs, the program runs either slow\_turn\_left() (line 148) function or slow\_turn\_right() (line 114) function accordingly.

**Crossings and stop**

After finishing up the curves and testing the code multiple times, the next obstacle in our path was crossings. We started working on the 90 degree crossing first. We created a function called all\_sensor (line 306), which activates as soon as all five sensors detect black and then it runs a timer for a certain time such that it can cross that 90 degree crossing. This function opens ups with the timer uses while loop to set the timer to 0.12 seconds and goes straight forward using straight\_fwd() function. Then it checks the sensors, and if all sensors are still black then it starts a while loop (line 332) to stop the robot using set\_motor\_speed(left, stop, 0) and set\_motor\_speed(right, stop, 0) functions. We had to adjust timer in line 313 multiple times to get the desire result but eventually it started working. However, while it was working we saw inconsistent results because sometimes it would work perfectly and other times it would not. Therefore, we decided to use the slow motion camera to see what was going on and why it would not work. We ran the robot multiple times until it made errors and that’s when we realized that sometimes the robot would not detect all five at once after going straight. When robot goes through a straight path and a crossing shows up, the robot would activate either left or right 90 degree functions because as the robot goes through the crossing sometimes a 90 degree left or right condition gets activated and it gets stuck into that function.

To counter this problem we added this condition in both 90 degree functions that if at any point in that function the robot sense all sensors then break out of the function as shown in line 198 for the 90 degree right turn and line 243 in 90 degree left turn. We also knew that we would run into a similar condition when the stop obstacle comes so we added a condition that only works if the five LEDs are still lit up after the crossing time. If five LEDs are still lit up after the crossing time the robot stops. It does not move until it senses something else other than the five sensors. Initially, we tried stopping by using the stop functions for both motors but the wheels had inertia, which kept messing up the stop so we had to add another reverse function such as , set\_motor\_speed(left, rev\_medium, 0); set\_motor\_speed(right, rev\_medium, 0) , in the stop part as shown in line 330 and 331. This made the wheels turn back for a very little time reducing inertia in the wheels and thus our stop worked better after that reverse function.

The next obstacle was the crossing at an angle. There was only one problem in the whole obstacle is that when our robot comes across a crossing at an angle it will activate our acute angle function and stays in that function until it senses white and that completely messed up our crossing. These conditions are in line 59 and 60, and 66 and 67. Therefore, we had to come with another condition that would happen after the acute angle gets activated. Now, we were able to differentiate between the crossing and the turn. We used the slow motion camera but our observation was different and wasn’t consistent so we decided to use the timers instead. In our acute angle function there was a condition, which starts a timer (line 347 and line 390), and if it did not find white for a certain amount of time, it would break out of that function using a return statement. After adjusting the timer a couple of times, our crossing started to work very well. Our timers were adjusted to .64 s as show in line 362 and 404.

**Merging and close track**

Now we turned our attention to the merging from gaps. Our gaps were working well and our merging in to a 90 degree turn worked very well. The only problem we were facing was merging at an angle. Our robot would go through the gap and as soon as it will hit the merging, the robot would turn the other way. We used our slow motion camera to detect what was going on and we realized that as the robot comes close to the merging, our 90 degree function gets activated since this condition in line 40 or 51 is met based on which way the merging is. Once this is condition is met, the robot acts like as if it were taking a 90 degree turn so to overcome that issue we added a condition in both 90 degree functions as shown in line 194 and 247, and if this condition is met then the robot gets out of this function. We added this condition because as the robot goes through the merging at an angle, it senses one of the two 90 degree functions and just before crossing the merge it senses the opposite 90 degree functions. Therefore, we programmed it to cancel the initial 90 degree function if this situation comes across, and since it only does in merging, our merging seemed to work well.

Our final obstacle was that robot should stay in the existing track even if it detects a nearby track. Again we used our slow motion to understand the possible conditions that occur when two tracks are nearby. We realized that a similar condition occurs such as in line 59 or 66 depending on which side is the nearby track. This condition calls for our turn\_30\_right() or turn\_30\_left() functions and our robot thinks an acute turn is coming, but after robot goes through another condition such as in line 63 or 70. So the way we programmed our code is that if this happens during the turn\_30\_right() or turn\_30\_left() functions then the function will return as shown in lines 356 and 398. Once the function returns the conditions on line 63 and 70 calls for the slow\_turn\_left() or slow\_turn\_right() function to correct the path of the robot as shown in lines 64 and 71. After multiple trials and success we concluded that our robot is working on these conditions. We also tried over and over to see if we find any errors, the only thing we realized is that deviation keeps changing so we had to adjust that constantly. It especially gets affected if we take out batteries and put them back again.

**Results – Track Challenge:**

We had three trials to get through a track on the final test day for the robot. On our first trial our robot performed quite well. However, the only problem it encountered was the merging. Our robot would not merge even after multiple attempts, and it did that at two places. After completing the first trial, we quickly designed a similar merging condition on our practice track and tried to use the slow motion camera to test our robot. We realized that our robot would detect the very end sensors such as in line 32 or 34 and then it will turn towards that side and once it does then it will meet this condition on line 40 or 51 which calls the turn\_90\_right() or turn\_90\_left(), and once it does then it will move forward and sense all black which will return the function as shown in lines 198 and 243. We tried to fix it in our gap function (whether\_gap() line 275) by adding conditions such as 0b00001 to turn left and 0b10000 to turn right but that would not work. It behaved the same way. We ran out of time so we had to go again, and this time it did the same exact thing but we had even more touches than last time so quickly went back and tried to tweak the code one more time. We thought about it a lot, and tried tweaking the whether\_gap() ,on line 275, function further. It didn’t result in any form of success. Then it suddenly hit us that once all black sensor lights up, it goes to the all sensor function, therefore; we should try to tweak that and see if it works. We made changes on all\_sensor() function as shown on line 315 to 322 on the basis that when the robot leaves the merging black line, whichever light was the last lit one, the robot would turn that way. We finally tried it on our own trial track multiple times and it worked. Finally our last turn came and we went for it. Our last trial was very successful and it made all the merges very easily. We got zero touches and we completed the whole track and every single obstacle in 40 seconds.

**Improvements:**

Our last trial run was very successful but we still think we could improve our speed. Especially when our robot goes through curves it felt a bit jittery as compared to others. If we play around with the speed of both wheels, maybe we can make it smoother on the track. It’s very clear that we can definitely improve the speed since there are couples of groups that have done the competition in shorter time than us. So it clearly indicates that there is a room for improvement here. We think we can improve our merging at an angle. It does work at this time but I would still like to try on few more times just to see if there are any glitches or any condition we are missing. More trials are more beneficial since we get to see conditions that we might have missed otherwise.