**Assignment 3 – Capturing Target Turtles**

**Due: 4/10/17 Time: 11:55pm**

**a) Team Name: Cyberdyne Industries**

Stephen Chan, stephen.chan9@yahoo.com (25% contribution)

Reyniel Maglian, renrenmaglian@gmail.com (25% contribution)

Vincent Campbell, vrcampbell@csu.fullerton.edu (25% contribution)

Mohammad Hassnain, mohammed-17@csu.fullerton.edu (25% contribution)

**b) Description of the heuristic strategy used to accomplish our goal:**

* Our heuristic strategy is to use a Best-First search approach. We reference our array of target turtles where the x and y coordinates are stored. An array is created to contain the distances of the target turtles from the original turtle. Our algorithm will look through the array of distances and move to that turtle via the moveGoal function. After moving to that first closest target turtle, turtle1 will reevaluate the distances to the remaining target turtles in the game space, it will then move to the next closest target turtle. This algorithm will run until there are no more target turtles left in the game space. If there happens to be a villain turtle in the path of turtle1 to a target turtle the associated distance cost with that target turtle will increase to avoid choosing the path that will get us killed. Lastly if there is no direct path that will not get turtle1 killed the algorithm will select the path with the shortest distance.
* The moveGoal function will try to direct the turtle around the villain turtle using the Distance\_tolerance variable as a foundation for how far away turtle1 has to stay away from the villain turtle. When the turtle comes into distance of villain turtle, the turtle will rotate to another direction and move a short distance in order to keep away from a villain turtle.

**c) Pseudocode for Heuristic in hwk3.cpp:**

**//Initialize**

initialize ros turtles

Create array for storing positions of 7 turtles.

Define a struct detailing each turtle’s position and name

TurtleClass tu[7]; // turtle struct object

SpawnName[][5] = {"T1","T2","T3","X1","X2","X3","X4"}; // turtle target and villian names

for(every turtle)

{

if ( i equals first or second index)

{

// Random location generator

x = random number

y = random number

// saving values in struct

tu[i].x\_point = x ;

tu[i].y\_point = y ;

Copy from x to str

Copy from y to str2

// Spawn turtle command for each turtle

strcpy(command, "rosservice call /spawn ");

strcat(command, str); // x coordinate

strcat(command, " ");

strcat(command, str2); // y coordinate

strcat(command, " ");

strcat(command, "0 "); // theta rotation

strcat(command,SpawnName[i]);

tu[i].Sname = SpawnName[i];

Call system sCommand

}

if(i>1)

{

tell = false;

x = random number

y = random number

}

// Check turtle spawn in same location

// also keep distance from other spawned turtles

k = 0;

// for(int k=0; k<i; k++)

while(tell != true )

{

if( (( x does not equal current k index x\_point ) && ( y does not equal current k index y\_point )))

{

if((k does not equal i) && (k equals (i-1)) )

{

// saving values in struct

tu[i].x\_point = x ;

tu[i].y\_point = y ;

Copy from x to str

Copy from y to str2

// Spawn turtle command for each turtle

strcpy(command, "rosservice call /spawn ");

strcat(command, str); // x coordinate

strcat(command, " ");

strcat(command, str2); // y coordinate

strcat(command, " ");

strcat(command, "0 "); // theta rotation

strcat(command,SpawnName[i]);

tu[i].Sname = SpawnName[i];

Call system sCommand

}

k++;

if(k equals ii)

{tell = true;}

}

else

{ x = random number

y = random number

}

}

}

int array\_of\_target\_distances[3]; // This array has the distance of each target from turtle initial point.

Call rotate(2.0 , call degrees2radians(45), 1);

**//Find the distance from starting point to each target and place it in array**

**// and use Heuristic Function to move to shortest distance spawn turtle**

double smallest\_target\_distance=0;

double total\_distance\_traveled=0.0;

int sum=0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// **Handles First Case**

for (u from 0 to number of target turtles)

{

array\_of\_target\_distances[u]= getDistance(turtle, goal\_turtle);

if(u is 0){

smallest\_distance=array\_of\_target\_distances[u];

}

else{

if (smallest\_distance>array\_of\_target\_distances[u]){

smallest\_distance=array\_of\_target\_distances[u];

}

}

}

sum = sum + smallest\_distance;

// save index to smallest distance turtle coordinates on first move

SindexDup = index\_smallest;

// Teleport turtle1 to shortest spawn turtle x ,y coordinates

strcpy(sCommand,"rosservice call /turtle1/teleport\_absolute ");

sprintf(str, "%d", tu[SindexDup].x\_point);

sprintf(str2, "%d", tu[SindexDup].y\_point);

strcat(sCommand, str); // x coordinate

strcat(sCommand, " ");

strcat(sCommand, str2); // y coordinate

strcat(sCommand, " ");

strcat(sCommand, "0.0"); // theta rotation

Call system sCommand

const char \*mychar1 = tu[SindexDup].Sname.c\_str();

Call system sCommand to kill spawn turtle

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// **Handles Second Case**

smallest\_distance=0;

index\_smallest=0;

for ((j from 0 to number of target turtles)

{

If (j does not equal SindexDup)

{

array\_of\_target\_distances[j]= getDistance(turtle, goal\_turtle);

}else

{

array\_of\_target\_distances[j] = 400; // reassign value of shortest distance already traveled to avoid repeat

}

if( j equals 0 ){

smallest\_distance=array\_of\_target\_distances[j];

}

else{

if (smallest\_distance>array\_of\_target\_distances[j]){

smallest\_distance=array\_of\_target\_distances[j];

index\_smallest=j;

}

}

}

sum = sum + smallest\_distance;

SindexDup2 = index\_smallest;

// Teleport turtle1 to shortest spawn turtle x ,y coordinates

strcpy(sCommand,"rosservice call /turtle1/teleport\_absolute ");

sprintf(str, "%d", tu[SindexDup2].x\_point);

sprintf(str2, "%d", tu[SindexDup2].y\_point);

strcat(sCommand, str); // x coordinate

strcat(sCommand, " ");

strcat(sCommand, str2); // y coordinate

strcat(sCommand, " ");

strcat(sCommand, "0.0"); // theta rotation

system(sCommand);

const char \*mychar2 = tu[SindexDup2].Sname.c\_str();

Call system sCommand to kill spawn turtle

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**// Final Handled Case**

for (g from 0 to number of target turtles)

{

if(g does not equal SindexDup2 && g does not equal SindexDup)

{

array\_of\_target\_distances[g]= getDistance(turtle, goal\_turtle);

}else

{

array\_of\_target\_distances[g] = 200; // reassign value of shortest distance already traveled to avoid repeat

}

if(g is 0){

smallest\_distance=array\_of\_target\_distances[g];

}

else{

if (smallest\_distance>array\_of\_target\_distances[g]){

smallest\_distance=array\_of\_target\_distances[g];

index\_smallest=g;

}

}

}

sum = sum + smallest\_distance;

SindexDup3 = index\_smallest;

// Move turtle to each target turtle

// strcpy(sCommand,"rosservice call /turtle1/teleport\_absolute 0 11 0.0");

// Teleport turtle1 to shortest spawn turtle x ,y coordinates

strcpy(sCommand,"rosservice call /turtle1/teleport\_absolute ");

sprintf(str, "%d", tu[SindexDup3].x\_point);

sprintf(str2, "%d", tu[SindexDup3].y\_point);

strcat(sCommand, str); // x coordinate

strcat(sCommand, " ");

strcat(sCommand, str2); // y coordinate

strcat(sCommand, " ");

strcat(sCommand, "0.0"); // theta rotation

system(sCommand);

const char \*mychar3 = tu[SindexDup3].Sname.c\_str();

Call system sCommand to kill spawn turtle

Print "Total Distance Traveled: " sum

/\*\*\***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//Function for checking if too close to a villain turtle**

void checkIfClose(TurtleSim::Pose turtlesim\_pose, TurtleClass turtles[]){

for (i from 3 to 7)

{

//Move the turtle before it gets close to .5 distance of a villain turtle.

if (getDistance(turtlesim\_pose.x,turtlesim\_pose.y,villain\_turtles[i].x\_point, villain\_turtles[i].y\_point) <=.10){

rotate(degrees2radians(90),degrees2radians(90), 1);

move(2.0,2.0,1);

}

}

}

**//Function for moving to a designated turtle.**

int moveGoal(turtlesim::Pose goal\_pose, double distance\_tolerance){

geometry\_msgs::Twist vel\_msg;

ros::Rate loop\_rate(10);

do{

//linear velocity

vel\_msg.linear.x = 1.5\*getDistance(turtlesim\_pose.x, turtlesim\_pose.y, goal\_pose.x, goal\_pose.y);

vel\_msg.linear.y = 0;

vel\_msg.linear.z = 0;

//angular velocity

vel\_msg.angular.x = 0;

vel\_msg.angular.y = 0;

vel\_msg.angular.z = 4\*(atan2(goal\_pose.y - turtlesim\_pose.y, goal\_pose.x - turtlesim\_pose.x)-turtlesim\_pose.theta);

velocity\_publisher.publish(vel\_msg);

ros::spinOnce();

loop\_rate.sleep();

} while(distance > distance\_tolerance);

cout<<"end move goal"<<endl;

vel\_msg.linear.x = 0;

vel\_msg.angular.z = 0;

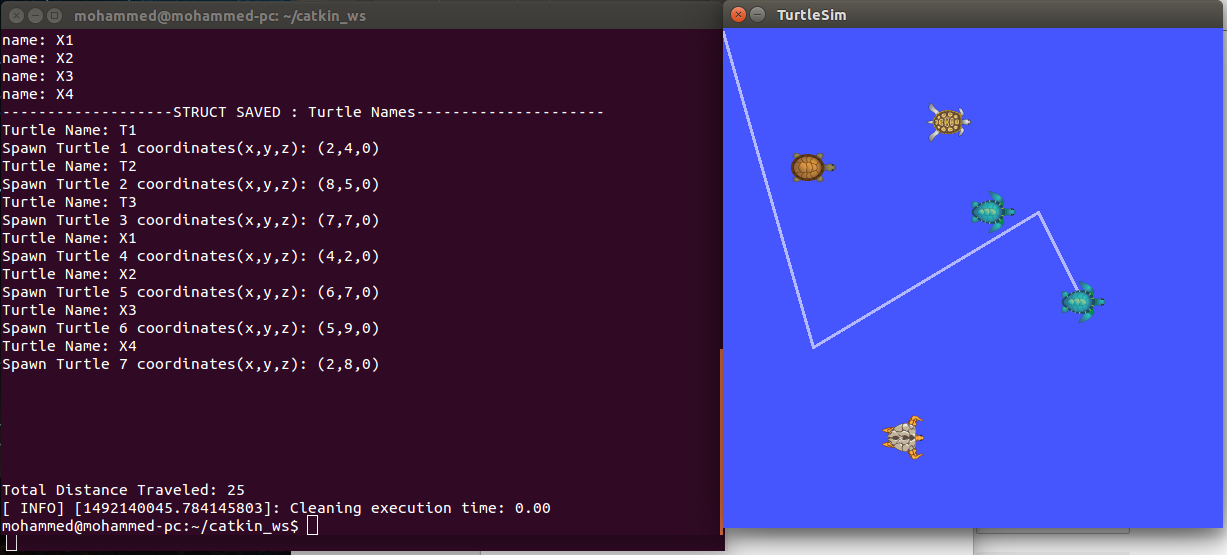
velocity\_publisher.publish(vel\_msg);

return distance

}

**Program screenshots:**

**Completed Run:**

****

**Resources Used:**

-Edu.gaitech.hk

<https://github.com/aniskoubaa/gaitech_edu/blob/master/src/turtlesim/cleaning_app/robot_cleaner.cpp> (code from the video)

**YouTube resources:**

ROS Tutorial 4.2: Moving in a Straight Line (Turtlesim Cleaner)

Covers robot\_cleaner.cpp setup and execution

<https://www.youtube.com/watch?v=PGZMlzBlMmw>

ROS Tutorial 4.3: Rotation Left/Right (Turtlesim Cleaner)

<https://www.youtube.com/watch?v=Ddqwq2WXFEk>

ROS Tutorial 4.4: Go-To-Goal Location (Turtlesim Cleaner)

<https://www.youtube.com/watch?v=Qh15Nol5htM>