CS3105 Artificial Intelligence Practical 1 AI in Action ID: 150007637

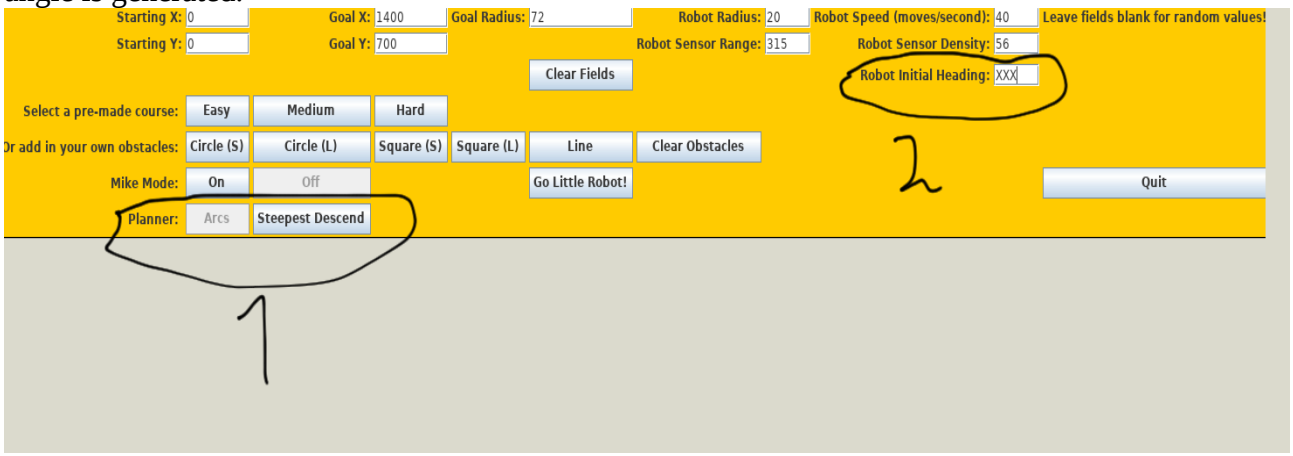
Legend:
R – robot centre
S – sample point
O – arc circle centre
M – move point on arc at a set fraction
G – goal point
V - 2nd and 3rd arc connection point (3-arc planner)

Design & Implementation

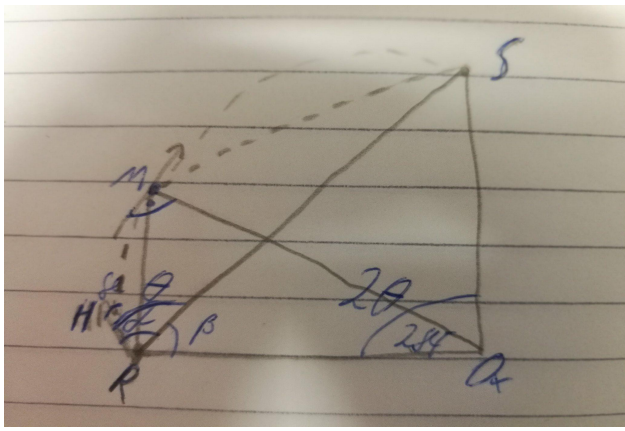
Note: I am going to roughly follow an approach where I pick up my major implemented pieces functionality and explain what has gone as expected and what issues I faced and could not find a suitable solution in time, while relating everything in a coherent manner.

GUI updates:

1. Planner: Fractional progress (Arcs) or Forwards Steepest Descend (greyed out means it is used now). Arcs planner is used now. **Default is forwards steepest descend.**
2. Robot Initial Heading: enter the initial heading angle of the Robot. If nothing is entered a random angle is generated.



Arc constructor (Arc class):



1. Calculate RS (sample vector) and RH (heading vector always 1)
2. Calculate the arc cos angle θ (HRS) by finding its cos from the ratio of the dot product and heading vector * sample vector and applying arccos on the result. Another way of finding it would be by first finding angle beta (ORS) and then subtracting it from the heading angle (HRO or alpha).
3. Perform a check on theta's angle and update appropriately if needed
4. Calculate total arc curvature and length.
5. Calculate move ratio RM / RS (set fraction 10 pixels for the robot to move through the arc).
6. Calculate half of the fractional turn angle gamma from the equation 2 * gamma / 2 * θ = move ratio (whole arc turn is 2θ ,explained for the fractional cae in step 7).

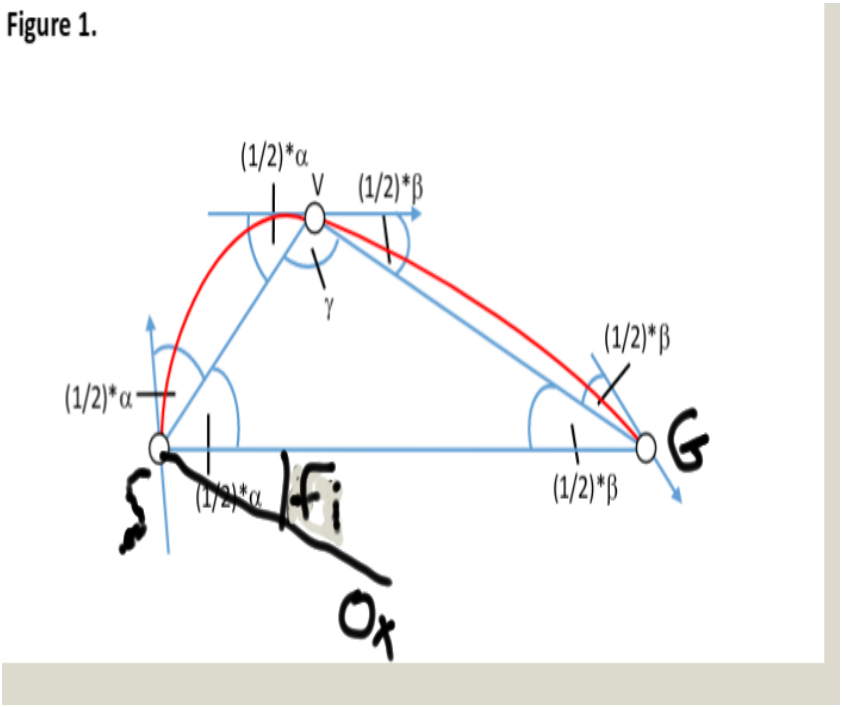
7. Update robot heading for the set fraction by 2 * gamma, one gamma to start going through arc , second time because the tangent line (robot heading vector at this fraction of the arc) with point M also introduces an angle gamma to complete the turn of the robot at that fraction.

3 Arcs path issue and explanation:

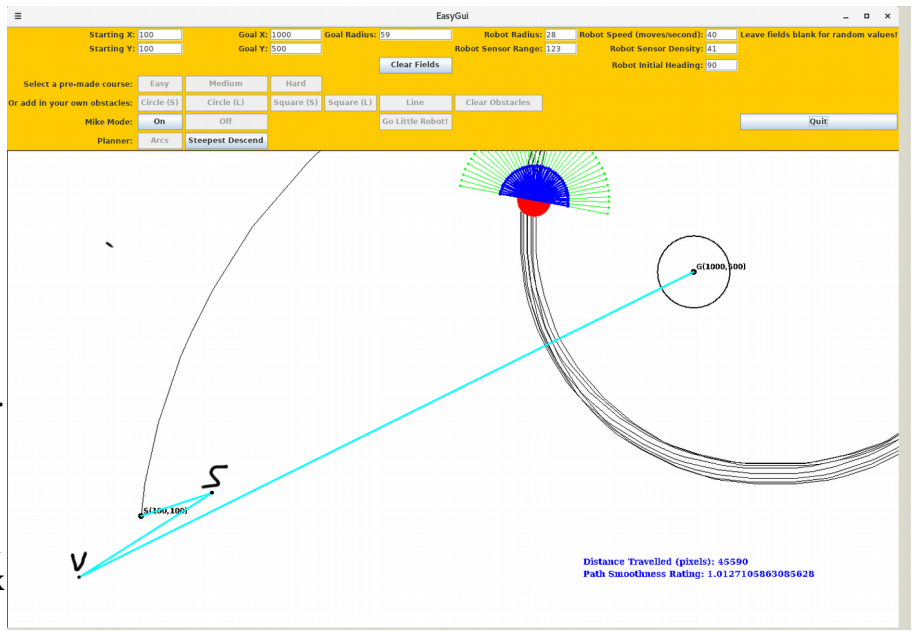
IsDone – not fully implemented but is supposed to check if the robot has finished its path over the current arc. In other words if the turn done over the current arc is equal to 2 * arc.getTheta.

GetTwoMoreArcs – This function is supposed to create the 2nd and 3rd arcs given a sample and goal points and robot heading at the sample point. It computes the angle “fi” between the goal vector and the Ox axis. Angle Fi needs to be subtracted from the robot heading at the sample point to get the total arc turn. An attempt is made to compute the point V, which is the end point of the 2nd arc and subsequently construct the 2nd and 3rd arcs.

Figure 1.



GetV – if alpha (angle between robot heading vector at sample point and the goal) is zero that means V and S lie on the same vector and no turn is needed to calculate point V. Otherwise call the calculateArcIntersection function.
Note: As shown in the screen shot below this function is not fully working. While, I have speculations why it may be, I could not get it to work in the required time. The robot is moving in wrong direction because the function isDone is not fully implemented, however, the problem is in calculating the point V. The blue lines show the three chords of the three arcs. Chord SV is supposed to point in the opposite direction and maybe a wrong calculation of angle “fi” is causing that problem or a problem in the calculateArcIntersection function.



CalculateArcIntersection – apply equations from the provided arc construction pdf, sin and cos theorem to calculate the x and y values of V. Given more time I would debug that and get the 3-arch planner working.

calculateHeadingArcs function (PotentialFieldsRobot class):
First, it gets an IntPoint from the robot's sensors (a light blue line is drawn, which is the chord of the sensed arc. It shows the development of the planned path).This function is used to replace the basic calculateHeading function used for the forwards steepest descent planner. It Updates the robot heading for the set fraction travelled by 2 * gamma, one gamma to start going through arc , second time gama because the tangent line (robot heading vector at this fraction of the arc) with point M also introduces an angle gamma to complete the turn of the robot at that fraction. It rarely travels through the whole length of the arc but only through a fraction. It computes a new arc when the robot's sensor finds a better arc to move through. This is done because I did not have time to implement a better method for surpassing obstacles and this manages to deal with the easy and medium difficulties but does not manage to overcome the C shapes. Making a new arc when a better arc is found provides advantage in time/space complexity and navigation around obstacles. However, moving through one arc through its whole length while fractionally increasing the turn of the robot and then starting a new one when the robot's goes through the whole turn of the arc (theta) would result in a more fluid movement but I have found my current implementation more advantageous because of obstacle navigation and time/space complexity. However, the 3-arc planner, which I did not manage to debug and fully implement would be the the best choice for most fluid movement of the three planners discussed, which would better cater for a real two wheel robot.
Note: This function also includes an extended version, but it is commented out because I could not manage to get it fully working in the limited time. The extended version is supposed to use a 3-arc planning system to navigate the robot (sample, first and second). The logic of the extension is: If AI is moving, it has already generated the three arcs and executes movement on them in consequent manner.

EvaluateSensorPoints – evaluate best sensor-able point in terms of the distance from the goal. I have used this method to use the seor points in stnsead of move points in order to create smoother arcs along the path because of the limited range of the moveable points.

Part 1 Comparison (Arc based fractional progress vs steepest gradient descent)

From the first sentence I understand that we have to say which way of sampling points and choosing where to go is better for both methods. Using sensor points is better for arc construction because the range is greater, so allows for an arc with a better turn and thus smoother path. For steepest gradient descent evaluating sample points directly passing the best to the move points evaluator is better because it allows for more rapid changes of direction, thus better progress overall.

Free space:
number of moves made:

Easy (no obstacles): forwards steepest descent

Distance(pixels)	Path Smoothness	Moves made	Angle	Sensor density	Sensor range	Robot radius	Goal radius
1940	0.050328408743 14408	97	0	20	233	30	70
1940	0.050328408743 14408	97	90	20	233	30	70
1940	0.046919148883 61204	97	180	20	233	30	70
1940	0.074987429936 39164	97	270	20	233	30	70

Easy(no obstacles): fractional progress

Distance(pixels)	Path Smoothness	Moves made	Angle	Sensor density	Sensor range	Robot radius	Goal radius
1940	0.255179354474 09995	97	0	20	233	30	70
1980	0.510097729875 7189	99	90	20	233	30	70
2540	1.028385466869 0449	127	180	20	233	30	70
2180	0.902607312015 3698	109	270	20	233	30	70

Brief analysis:
We can see that when the start angle is bigger, an arc with more turn has to be made, so the fractional progress is worse than forwards steepest descent. However, when angle is 0 it shows much smoother path, with flat arcs. The first also shows better consistency as expected because its starting angle does not matter as not having to do an arc based path.
Note: The path smoothness not being better in the second planner in general is due to the fact that, although the robot moves in arcs, underneath it still draws small separate lines.

Extended comparison:

Easy (obstacle present): forwards steepest descent

Distance(pixels)	Path Smoothness	Moves made	Angle	Sensor density	Sensor range	Robot radius	Goal radius
2060	0.6415398901240176	103	0	20	233	30	70
2060	0.6488766747459759	103	90	20	233	30	70
2060	0.04691914888361204	103	180	20	233	30	70
1940	0.6348244328868484	97	270	20	233	30	70

Easy(no obstacle present): fractional progress

Distance(pixels)	Path Smoothness	Moves made	Angle	Sensor density	Sensor range	Robot radius	Goal radius
2120	0.8278430342015479	106	0	20	233	30	70
2180	0.9701842935060843	109	90	20	233	30	70
2600	1.199169851814865	130	180	20	233	30	70
2260	1.14519081998186	113	270	20	233	30	70

Brief analysis:

The first stays consistent as we introduce an obstacle. As we introduce an obstacle the second planner’s path is increased significantly because it has to construct arcs around it.

Extended comparison:

Medium: forwards steepest descent

Distance(pixels)	Path Smoothness	Moves made	Angle	Sensor density	Sensor range	Robot radius	Goal radius
3480	0.6045627628189228	174	0	20	233	30	70
3560	0.6488766747459759	178	90	20	233	30	70
3520	0.7826480301046939	176	180	20	233	30	70
3480	0.6045627628189228	174	270	20	233	30	70

Medium: fractional progress

Distance(pixels)	Path Smoothness	Moves made	Angle	Sensor density	Sensor range	Robot radius	Goal radius
4720	1.4691647402287458	236	0	20	233	30	70
4480	1.6990820721335829	224	90	20	233	30	70
4220	0.9885761877731458	211	180	20	233	30	70
4640	1.586529598528873	232	270	20	233	30	70

Brief analysis:
We see more difference in the distance travelled in the steepest descent planner due to the differences of path selected around obstacles. The arcs approach did not go between the obstacle coarse at all at 180 degrees, because it just rotates over it. While, the forwards steepest descend outperforms the fractional progress planner, the latter looks much more fluid and has better scalability for future development.

Note: Overall testing analysis of the fractional progress planner shows that the robot performs poorly with obsticles when sensor range is more than about 400-500, because it makes huge arcs, thus harder to escape obsticles. It also performs poorly with sensor density of less than 4 as expected.

Evaluation & Conclusion

Overall I think that I have made a reasonable attempt at this practical, providing First part implementation, with almost working 3-arc planner, but fully functional 1-arc planner. However, if given more time I would have focused on avoiding C shaped obstacles by the winding/unwinding principle.

Testing

I have performed various tests, with custom obsticle tracks and easy/medium/hard tests. I have covered corner cases and am confident that my 1-arc planner deals with Medium difficulty courses.

References:

http://rossum.sourceforge.net/papers/CalculationsForRobotics/CirclePath.htm
http://ieeexplore.ieee.org/document/5509519/
https://stackoverflow.com/questions/2150050/finding-signed-angle-between-vectors