CSE Special Project Report

Push Recovery through Ankle Adjustment

Tianjiu Yin
yintianjiu@gmail.com
special project A

November 2012

School of Computer Science & Engineering
University of New South Wales
Sydney 2052, Australia

Supervisor
Bernhard Hengst
K17 401E/ +61 2 9385 56993

Acknowledgements

Firstly, and most gratefully, thank you, Bernhard Hengst for your supervision in this project and great patience for listening and explaining to me all the time. Without your help, I definitely would not learn so much and finish this project.

Secondly, thank you, Sean Harris, Richard Hua, Youssef Hunter, Belinda Teh for always answering my questions warmly.

Finally, thanks to rUNSWift team and CSE school for providing me with such a unique experience in Sydney and I really enjoy the this fantastic semester.

Contents

Abstract

This report explores the issue of push recovery using only the adjustment of ankles. At first, we discuss about the physical model of the robot and use physics to simulate the movement of robot when shaking. Then, we adopt the reinforcement learning as the main method to generate adjusting strategy and verify the correctness this method. Finally, we try to implement the policy on the robot. Also, we discuss about the limitations and solutions of the method used in this project.

1 Introduction

1.1 Motivation

According to the video of semi-final competition, Robocup 2012[1], our robots fell down 14 times and 11 of them were in the first half while in the first half, the opponent robots only fell down 4 times. In fact, in most cases, the robots fell down due to slight collision or stepping on other robot's feet. However, without push recovery, it is highly possible that the robot would toggle if the robot tilts more than 10 degrees since the back part of the robot's foot is comparatively short and max degree of tilting backwards is around 13 degrees. In the experiment, if the robot was pushed 10 degrees backwards, it would still toggle after fluctuating several times since it has a longer forward part of foot. Thus, for a stiff robot like a wood pole, you cannot expect the robot to sustain too much push without losing balance.

However, for human beings, it's natural and easy to make small and limited adjustments of different part of the body to prevent the body toggling against a push. Thus, same for robot, if would can make the robot more sustainable to the push, not only can we save the time falling down and getting up, we can also take advantage of keeping position of the ball firmly, which would be definitely an advantage for football competition.

Moreover, increasing the ability of keeping balanced can also contribute to more advanced walking and more sophisticated shooting or passing. For example, if the robot is sustainable enough, the robot may shoot while walking and tilt more degrees within a short time and with more power like a real football player rather than settle down first and lift one foot slowly.

All of above are expectations of techniques far beyond what we have right now, but we are starting from the very basic part of push recovery to make it possible one day.

1.2 Aims

For me, the goals for this project are as follows:

1. Build the physical model of the robot and work out the transition function.
2. Find out whether the ankle adjustments would actually work to help robot recover from a push.
3. Generate the policy using reinforcement learning.
4. Implement the policy on the robot and find the problems in practice.

1.3 Outline

Chapter 2: this section describes the background of this project, providing the related research

outcomes and previous work done by other teammates.

Chapter 3: this section describes all the assumptions and measurements that this project is based on.

Chapter 4: this section describes the physical model of the robot and velocity and angle equation when robot is shaking.

Chapter 5: this section describes how reinforcement learning is adopted in this project and how to generate the policy in the end.

Chapter 6: this section describes some improvements based on the discussion of chapter 5 to make the results more accurate.

Chapter 7: this section describes how to implement the results in the robot and problems concerning with practical implementations.

Chapter 8: this section describes the main problem encountered in this project and some possible solutions.

Chapter 9: this section concludes and summaries the findings of this project.

2 Background

2.1 Related research

Actually, Bernhard had published a paper whose content is similar to this report but use a different way to generate policy [2]. There are also other researches that focus on push recovery of bipedal robot. Albertus attempted to modify the walking phase dynamically in order to keep the robot balanced and recover from a push [3]. John used the technique of learning capture points to compute the steps of robot to take [4].

2.2 Previous work

In the rUNSWift team, some related work was completed last year and there is a Brock branch in the rUNSWift github repository that implemented the policy based on the NAO 3 system. But in the actual competition, this control of body balance was not adopted. To make the sensors more accurate, they tried to compute the angle and velocity according to the vision of the robot and the results turned out fine.

3 Assumptions and measurements

3.1 Assumptions

Since in this project we only investigate the elementary part of the whole research, we make a lot of assumptions to simplify the problem, though the results may be crude, to make sure that this method works both in theory and basic situations. All the assumptions are listed as follows:

A. After a push, only ankles would rotate and other part of robot would stay stiff.
B. During a very short time, the CoM won't change and the change of actions only leads to the rotation of foot (the point of ankle stays static as well).
C. During a very short time, the foot would rotate while the body would rotate slightly.
D. The position of pivot follows a liner function.
E. The robot will change to certain action in zero time.
F. The robot touches ground made of carpet and its feet touch the ground totally permanently.
G. During a very short time, the pivot wouldn't change.

3.2 Measurements

No previous report shows the height of CoM and other figures. To start with, we first measured related figures of the robot especially the height of the CoM of the robot. The table below shows we need in this project.

Table 1 Measurements of robot

Name	Data
Standing height	549.00 mm
Standing height of CoM	262.36 mm
Foot length	162.00 mm
Back part of foot	57.15 mm
Forward part of foot	104.85 mm
Max degree of leaning backward	13.75°
Max degree of leaning forward	17.34°

4 Physical model

4.1 Force analysis

In order to better understand the way that a robot moves while shaking, it's necessary to analyze the force of robot accurately.

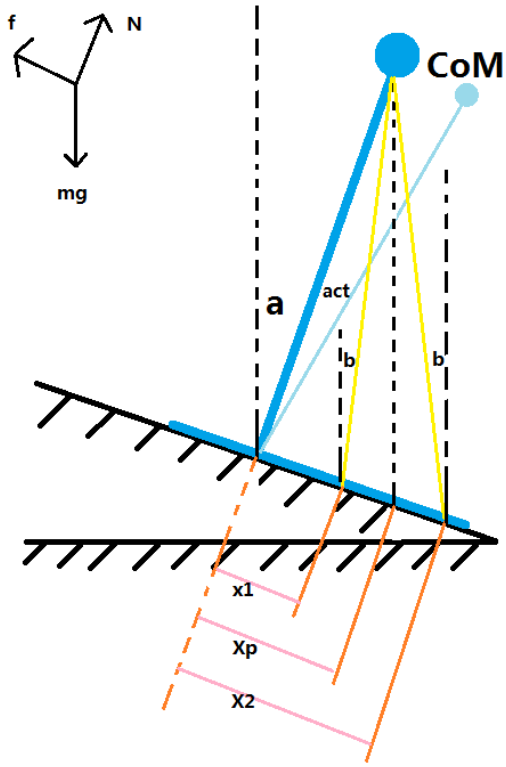

Figure 1 physical model

Table 2 representation of characters

name	Meaning
a	The angle between perpendicular line and the body (forward positive)
act	The rotation angle of ankle (forward positive)
b	The angle between perpendicular line and line between CoM and pivot
X_1, X_p, X_2	Distance between ankle and pivot (forward positive)
f	Fraction
N	Elasticity

mg	Gravity
CoM	Center of mass

From the figure we can see that the robot sustains gravity, fraction, elasticity and their direction respectively.

According to assumption F, in a very short time (0.01s) the rotation of ankle only leads to the slight change of CoM without the change of foot as we see in the figure.

4.2 Pivot function

According to assumption D, we consider the whole robot in this model as an inverted pendulum and we assume that the pivot of change within the range of foot linearly. In fact, we raised several functions and the following one works well:

$$X_p = \min\left\{l_f, max\{l_b, k_1(a - act) + \sin(act) \cdot l\}\right\}, \quad \text{Equation 1}$$

$$l, the\ length\ of\ foot$$
$$l_b, the\ length\ of\ back\ part\ of foot$$
$$l_f, the\ length\ of\ forward\ part\ foot$$
$$k_1, constant\ parameter$$

In this project, we set

$$k_1 = \frac{l_f}{M_a}, \quad \text{Equation 2}$$

$$M_a, the \max degrees\ of\ leaning\ foward$$

To make sure we confine the pivot position within the range of foot, we set the value of pivot to the length of forward or backward part of the foot when exceeding the boundary. Also, the k_1 is selected through experiment and works comparatively well in this project.

4.3. Angle acceleration

The key of this model is to compute the angle acceleration so that we can calculate the position of the robot after a short time. According to assumption B and G, both the positions CoM and pivot would not change, thus the angle acceleration would keep the same in a very short time.

According to Moment of Inertia,

$$mgh \cdot \cos\alpha = \frac{4}{3}mh^2 \cdot \beta \qquad \text{Equation 3}$$

gives

$$\beta = \frac{3g}{4h}\cos\alpha, \qquad \text{Equation 4}$$

$\alpha, angle\ b\ in\ the\ figure$
$\beta, angle\ acceleration$
$h, standing\ height\ of\ CoM$

5 Reinforcement Learning

5.1 Introduction to reinforcement learning

Reinforcement learning, inspired by behaviorist psychologist, is an area of machine learning in computer science, concerned with how an agent ought to take actions in an environment so as to maximize some notion of cumulative reward [5]. There are 4 key elements of reinforcement learning, a policy, a reward function, a value function, and a model of the environment respectively [6]:

A policy defines the learning agent's way of behaving at a given time.
A reward function defines the goal in a reinforcement learning problem.
A value function specifies what is good in the long run whereas a reward function indicates what is good in an immediate sense.
A model of the environment mimics the behavior of the environment.

In this chapter, we use reinforcement learning to generate the policy to make the robot keep balanced. In the chapter 4, we already construct a model of the environment and we focus on the policy generation.

As we know, the robot move continuously and so, there requires a continuous transition function. In this project, we simplify the problem that we divide the continuous angles, velocities into several preset value, namely a table. And Q table means the value of max reward in the each unit of this table.

5.2 Reward

The reward determines the goal of the reinforcement learning. In this project, we would like to make the robot settle down as fast as possible. Therefore, the goal should be that both angle and velocity equals to zero. To make goal easier to achieve and avoid the robot fluctuating forever, we increase the goal area. In this project, we set different goals areas and compare the difference of the result. Normally, the reward function is as follows:

$$reward(a,v) = \begin{cases} 1, & \{(a,v)|a \in (-2,2), v \in (-2,2)\} \\ 0, & otherwise' \end{cases} \qquad \text{Equation 5}$$

$$a, angle$$
$$v, velocity$$

It's unnecessary to make continuous reward since it is reinforcement learning that would learn the different rewards and its comparative value.

5.3 Transition function

In the chapter 4, we have already worked out the angle acceleration. It's simple to compute the next position according to the acceleration and velocity. However, the trick here is the angle we have in the chapter 4 is the angle between perpendicular line and line between CoM and pivot while the angle and velocity that can be measured through the sensors of robot are the angle between perpendicular line and its body. Thus, we have to transfer the angles and velocities under two ways of angle measurement.

$$\mathrm{l} = \sqrt{h^2 + {x_p}^2 - 2hx_p \cdot \cos\left(\frac{\pi}{2} - act\right)} \qquad \text{Equation 6}$$

$$\mathrm{b} = \mathrm{angle} - \frac{x_p}{|x_p|}\cos^{-1}\left(\frac{h^2+l^2-{x_p}^2}{2hl}\right) \qquad \text{Equation 7}$$

$$\mathrm{acc} = \frac{3h}{4l}\sin b \qquad \text{Equation 8}$$

$$\mathrm{v} = velocity + acc \cdot t \qquad \text{Equation 9}$$

$$\mathrm{b}' = b + velocity \cdot t + \frac{1}{2}acc \cdot t^2 \qquad \text{Equation 10}$$

$$\mathrm{a} = \frac{x_p}{|x_p|}\cos^{-1}\left(\frac{h^2+l^2-{x_p}^2}{2hl}\right) + b' \qquad \text{Equation 11}$$

t: a short time

$velocity, angle$: actual data from sensor

v, a: velocity and angle after time t

From equations above, we can predict the angle and velocity of the robot after time *t*. Although the frequency of our robot is $100\ H_z$, we set the t to a much smaller value like 0.001s and compute the sum of 10 movements to make the result more accurate.

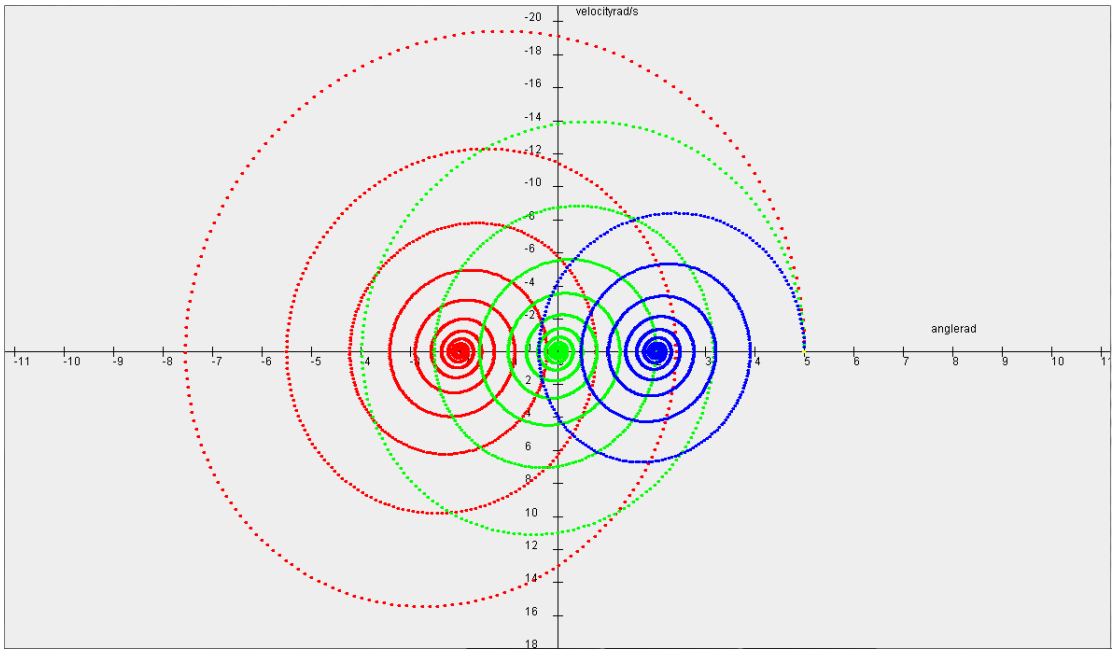

Figure 2 transition function

5.4 Q table

To get the Q table, we can run the transition function repeatedly until it's converged. Before we actually run the transition function, several things have to make clear:

Firstly, in the reality, there exist fraction and object would lose energy gradually. But in our previous equation, velocity would not damp and may increase due to the approximate numeric representation in the computer after several computations. So every time we run the transition function, we set a parameter of *damp* to make the result more reasonable:

$$velocity *= damp \qquad \text{Equation 12}$$

The value of damp may vary according to the value of time t.

Secondly, we need the set a parameter called step-size parameter ∂ to distinguish the steps each action taken. In this project, we use following equation to compute the max reward, namely the value of Q table:

$$value(angle, velocity)$$
$$= \max\left\{reward(angle, velocity) + \partial \cdot \max\left\{reward(angle_i, velocity_{i)}\right\}\right\},$$
$$angle_i, the\ angle\ after\ time\ t\ when\ taking\ action\ i$$
$$velocity_i, the\ velocity\ after\ time\ t\ when\ taking\ action\ i$$
$$i = 0, 1, 2, 3 \ldots \qquad \text{Equation 13}$$

To fill in the value of whole Q table, we only need to run this function for each unit in the table.

Thirdly, there is also another important factor of computing Q table, that is, the grid size. Obviously, the bigger the table is, the more accurate result and better decision we can make. However, on the one hand, it's computably impossible to increase the table to large. In practice, it would take more than an hour if you divide both angle and velocity into 600 parts with 3 actions. On the other hand, the angle and velocity change unevenly. When the velocity is quite large, the angle would change rapidly and it would be unnecessary if you divide them too many parts. Reversely, if both the velocity and angle are small, it would require surprisingly large table to differentiate two consecutive movements. There exist several solutions mentioned in the chapter 6.

5.5 Policy

Actually, we can generate the policy while generating Q table but it's a separate part here for a logical consideration. In the previous discussion, we didn't refer the important factor 'action'. In fact, action is a simple version of ankle rotation. To simply this question, we preset certain actions that represent different actions, for example we can set 3 actions $\{-2,0,2\}$ means that leaning backward -2 degrees do nothing and leaning forward 2 degrees rather than take continuous degrees of actions.

Thus, when we compute the value of Q table, we record the action that would yield max long term reward in a file. And when we actually run the robot and the robot would read from this file and select the optimistic action to take in order to help robot recovery from unbalance.

6 Further Improvements

6.1 Choice of actions taken

To save energy and reduce fraction, the robot should change fewest times to settle down. To achieve this goal, we can do following things:

Firstly, set the priority of actions. For example, we can present the priority of actions and choose the action with higher priority when several actions would generate same reward.

Secondly, set the cost of changing actions. Rather than purely calculate the cost of time, we can add the cost of changing actions. For example, each time we change the actions, we will only get 99% of the original reward to find out that weather the action change is worth enough. Furthermore, we can set various cost of action changing according to the degrees that would change from prior state.

6.2 Increase the grid size of the Q table

In this section, we draw several figures to show how the result would differ when change the grid size.

Table 2 corresponding value of figures

Figure No	Grid size (velocity, angle)
3	30, 30
4	60, 60
5	100, 100
6	300, 300
7	400, 400
8	500, 500
9	600, 600

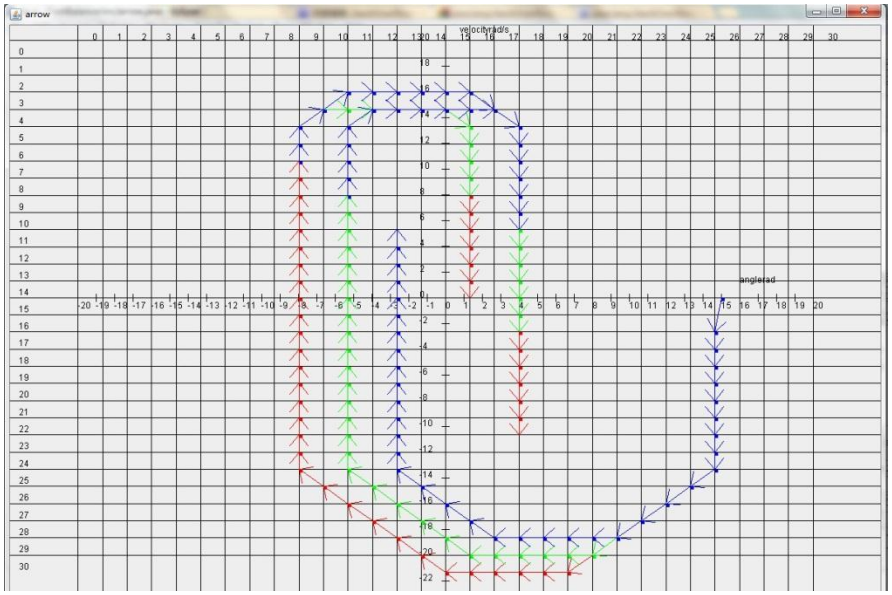

Figure 3 grid size changes 1

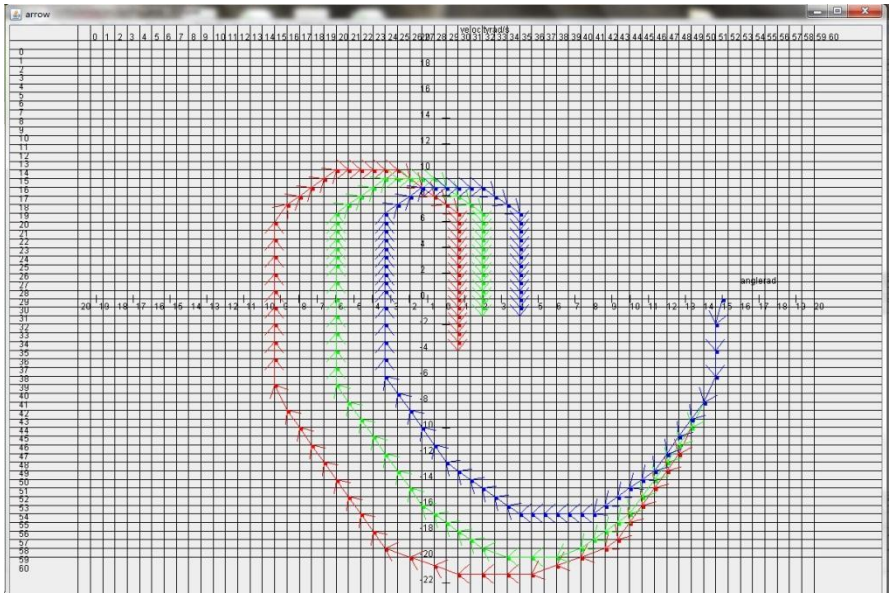

Figure 4 grid size changes 2

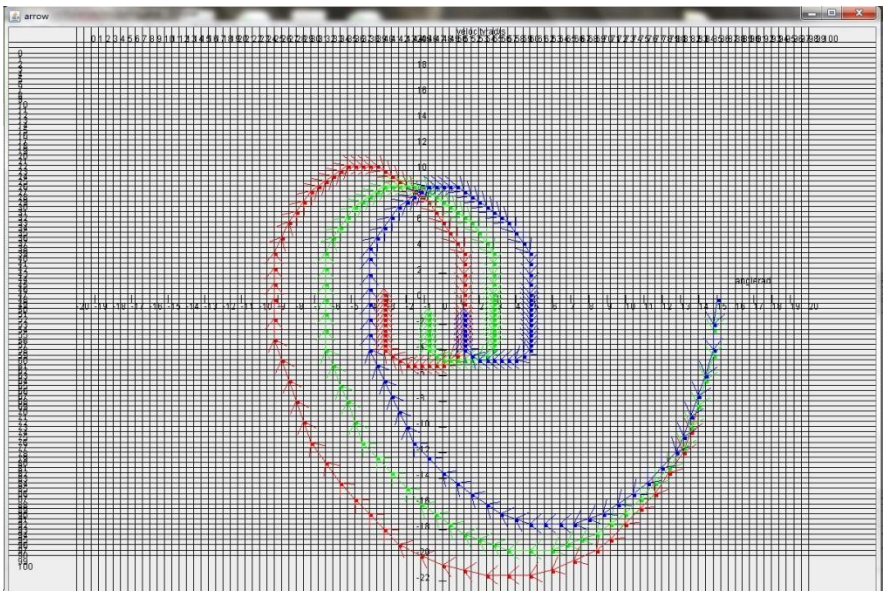

Figure 5 grid size changes 3

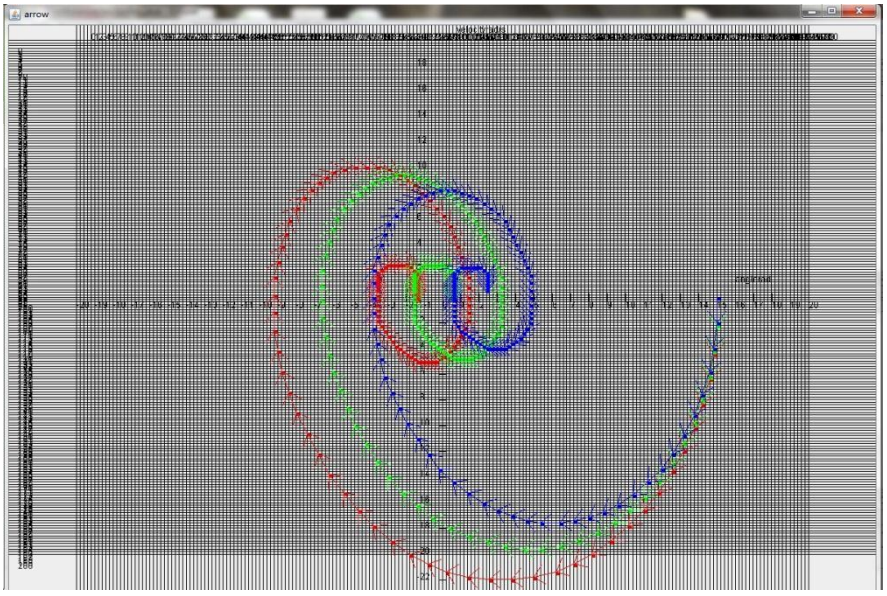

Figure 6 grid size changes 4

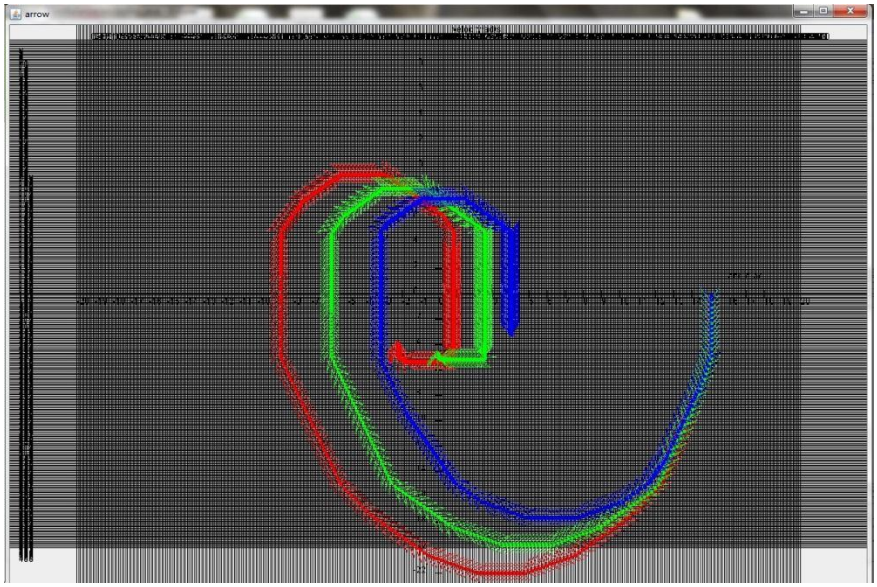

Figure 7 grid size changes 5

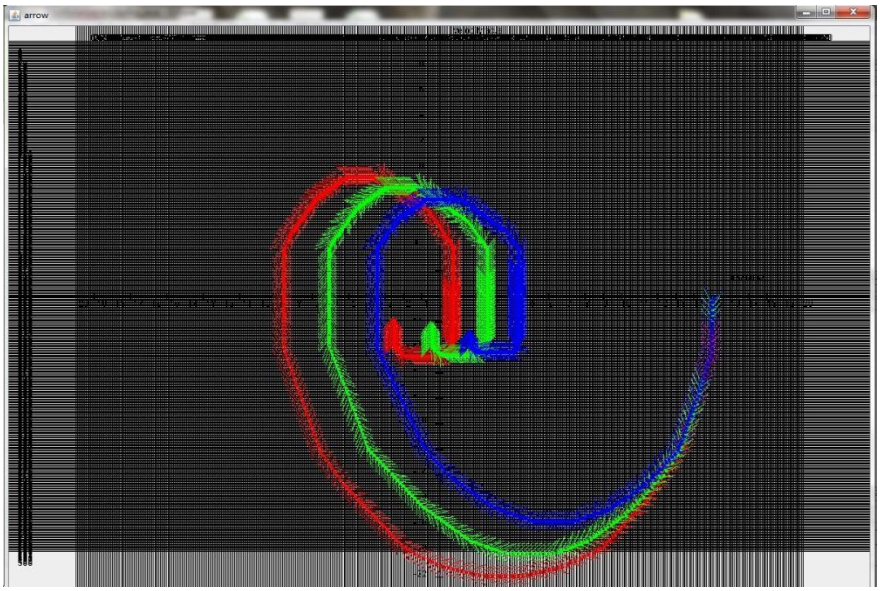

Figure 8 grid size changes 6

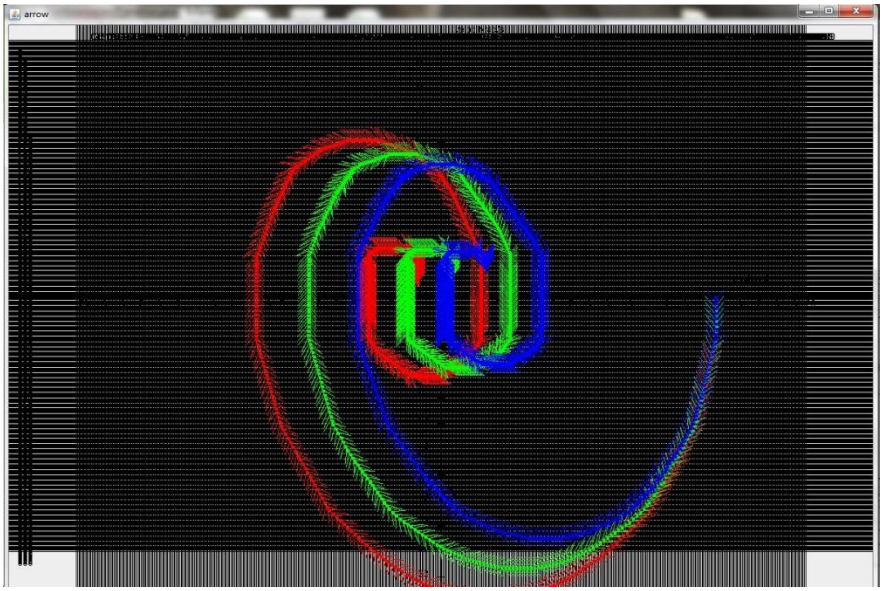

Figure 9 grid size changes 7

From those pictures, we can see that increasing the grid size would solve part of the problem but not the whole problem. Since we have discussed, the velocity and angle would change quite subtly when approaching the goal area. Therefore, it's not a reasonable way to work out the accurate result.

6.3 Variable grid size

Since increasing the grid purely would not solve the problem efficiently, variable grid size may be an alternative way to solve this problem, which means, in general, we set the grid size according to the value changing rate and if the value changes rapidly, we decrease the grid size and vice versa. The figure below shows the gist of this method.

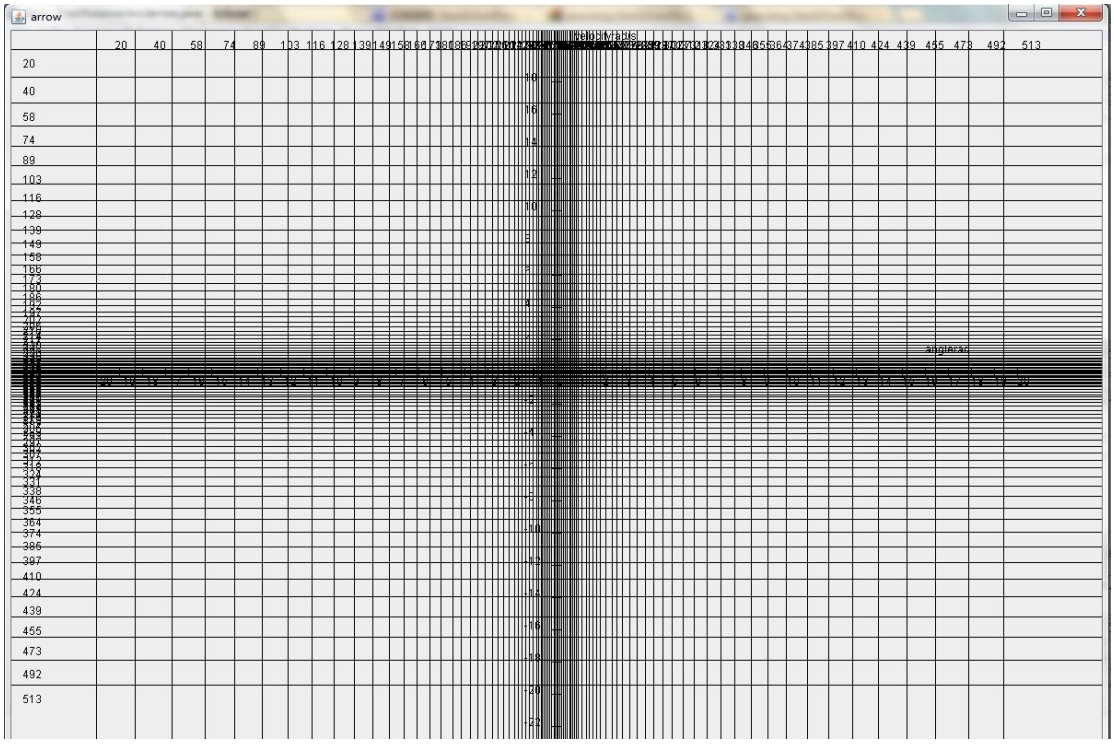

Figure 10 variable Q table

They are two problems in this solution:

Firstly, when approaching the axis, the Q table would become more dense and other way around. However, it breaks the basic principle of table or array. If we still use array as the data structure, there would be a lot of waste of memory and if we choose other data structure like links, it takes much more time to look up in the table.

Secondly, this method only alleviates the problem rather than solve it. On the one hand, there are still some areas near the goal area out of reach. On the other hand, it's hard to generate an optimistic variable table that would be smallest and contain all the policy we need.

Therefore, this method is only slightly better that the first solution.

6.4 A star

A star is an algorithm that is widely used in path finding and graph traversal, the process of plotting an efficiently traversable path between points, called nodes[7]. It perfectly solves the problems we have since it can compute with continuous angle and velocity and decide which the best action to take is. The basic function is as follows:

$$f(x) = g(x) + h(x) \qquad \text{Equation 14}$$

$$f(x)\ is\ the\ expected\ cost\ in\ total$$

$$g(x)\ is\ the\ actual\ cost\ so\ far$$

$$h(x)\ is\ the\ expected\ remaining\ cost$$

In this project, we can use 2-parameter function as follows:

$$f(v,a) = g(v,a) + h(v,a)$$
$$v\ is\ velocity\ and\ a\ is\ angle$$

We define the cost as the time cost. Thus $g(v,a)$ is simple and just equals the time so far. The core challenge here is to find heuristic function to predict the cost while never overestimate. It seems like we know the actually cost to some degree but not totally. One straightforward way is to compute the distance between the position and goal area, namely$\sqrt{v^2 + a^2}$. However, for one thing, the time and distance have different units and hard to compare them. Moreover, the distance here does not necessarily correspond to the actual cost since in the same distance, when it is approaching the axis, it would move slowly and take more time while the other way round.

In this project, we find an alternative way to solve this problem that we can use greedy algorithm to calculate the expected cost, which would compute the action each time that would generate next position closest to the goal area. The function is as follows:

$$h(v,a) = \begin{cases} h(v',a') + 1, & (v,a) within\ goal\ area \\ 1, & otherwise \end{cases}$$

where

$$\left\{(v',a') \middle| \sqrt{v'^2 + a'^2} = min\left\{\sqrt{{v_i}^2 + {a_i}^2}\right\}, i = 0,1,2,3 \ldots\right\}$$

$$v_i, a_i\ are\ the\ velocity\ and\ angle\ after\ time\ t\ and\ action\ i\ taken$$

Actually, this algorithm works well and could achieve the result even better than reinforcement learning. The figures below show the difference of different methods:

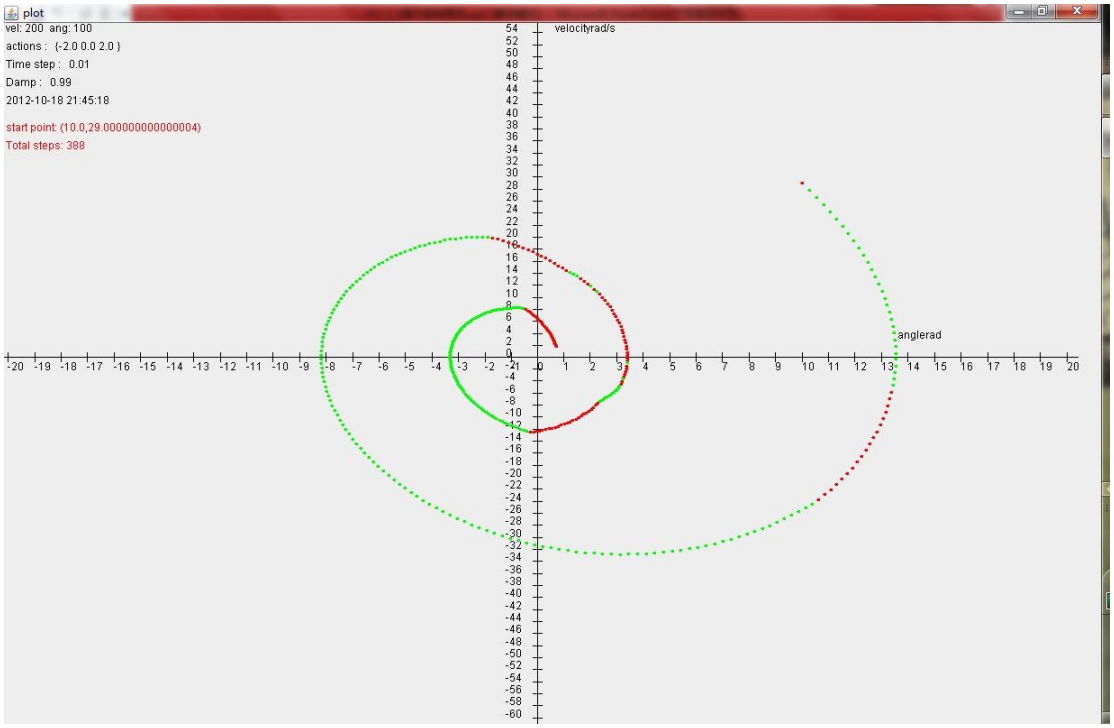

Figure 11 reinforcement learning

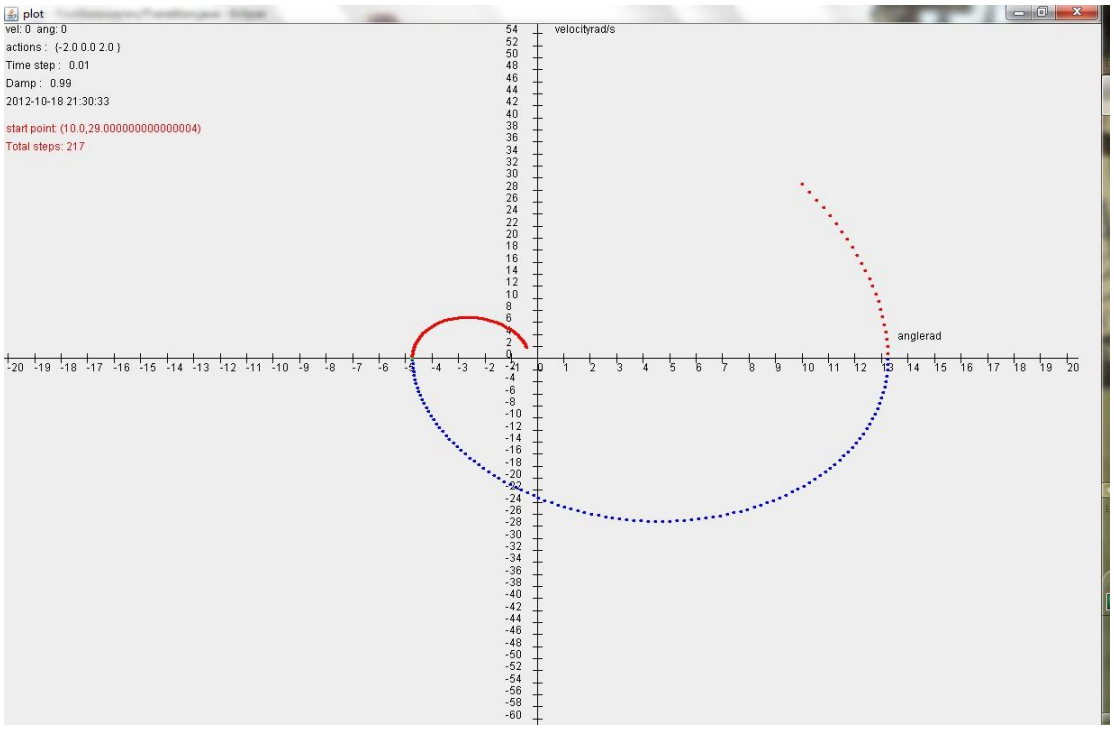

Figure 12 greedy algorithm

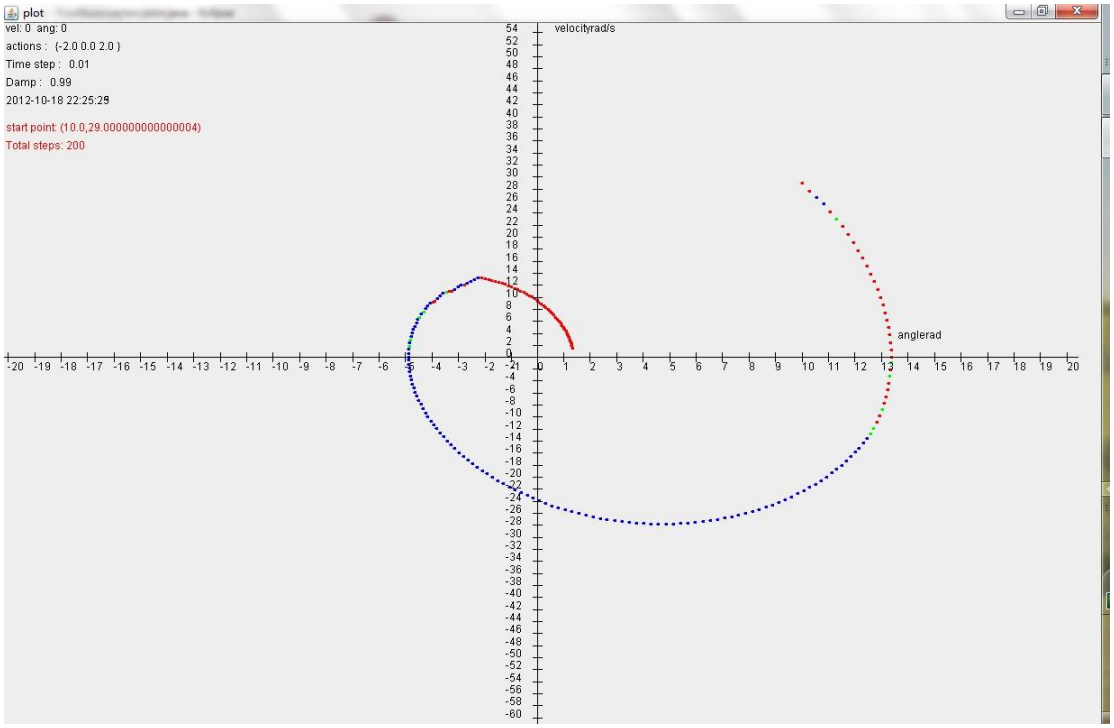

Figure 13 A star

More comparisons can be seen from the table:
Initial state:

Velocity= 30°/s
Angle= 10°
Damp rate= 0.99

Table 4 corresponding value in the figures

Method	Grid size (velocity, angle)	Time (0.01s)
Reinforcement learning	10, 10	427
	30, 30	427
	50, 50	427
	100, 100	411
	200, 100	388
Pure greedy algorithm		217
A star combined with greedy algorithm		200

From these figures and table we can see that A star work much better than reinforcement learning at this stage. However, to compute the expected cost this way every time is time-consuming. But we can compute the result ahead and store them in a file. Otherwise, there are still possible to increase the speed of this algorithm and if we can compute the best action within 0.01s so that the robot could make real-time decision without worrying about the problem of discrete representation of velocities and angles.

7 Implementations

7.1 rUNSWift Motion Architecture

Although there are detailed reports that explain the architecture of whole rUNSWift code, some more explanations of motion specifically here may help someone else who would like to understand how motion works quickly.

In general, there are 3 aspects of motions, namely touch, generator and effector. Touch focuses on collecting data of sensors from perception, effector concentrates on performing the actions that decided by generator. In this project, we implement our code in generator. Generally speaking, there are many different types of motions and each of them has a separate motion generator like StandGenerator. Some of them are purely decision-making visual files like RLPlanner.

7.2 Implementation

To simplify the implementation, we directly add modifications in the WalkEngine file. That is to firstly get the data of velocity and angle, and then look the best action to take from the policy file and finally directly set the angles of its ankles.

7.3 Difficulties

7.3.1 Noise of sensors

In fact, the angle sensor is comparatively accurate but velocity sensor is noisier. To test the accuracy of sensors, we record the data of sensors when robot standing still and given a push.

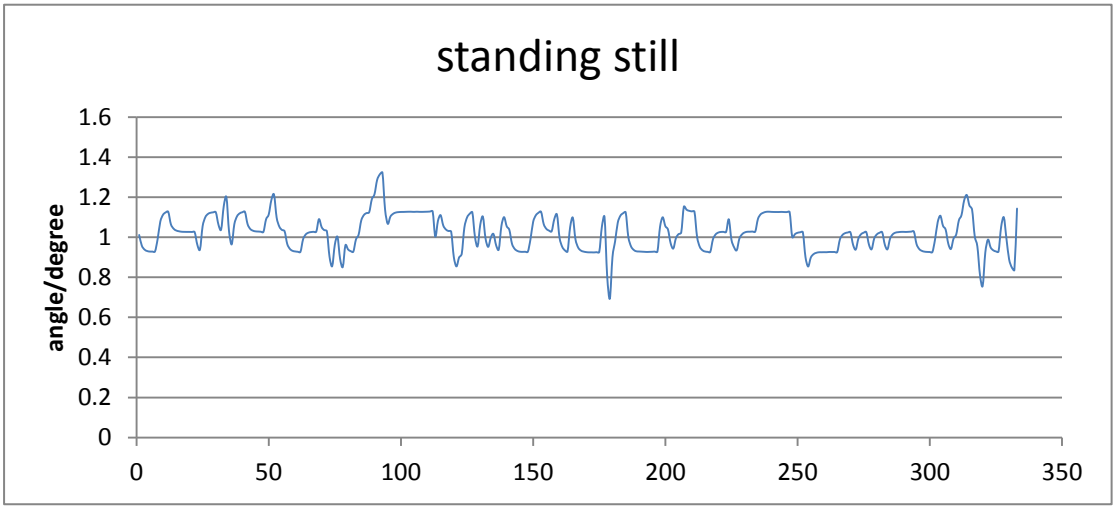

Figure 14 angle distribution while standing still

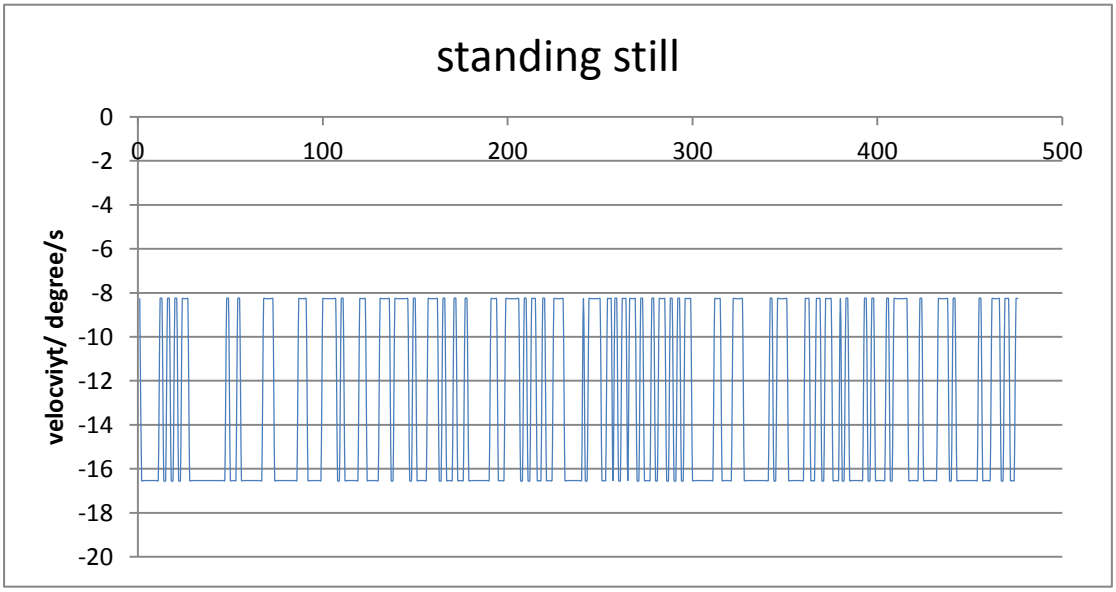

Figure 15 velocity distributions while standing still

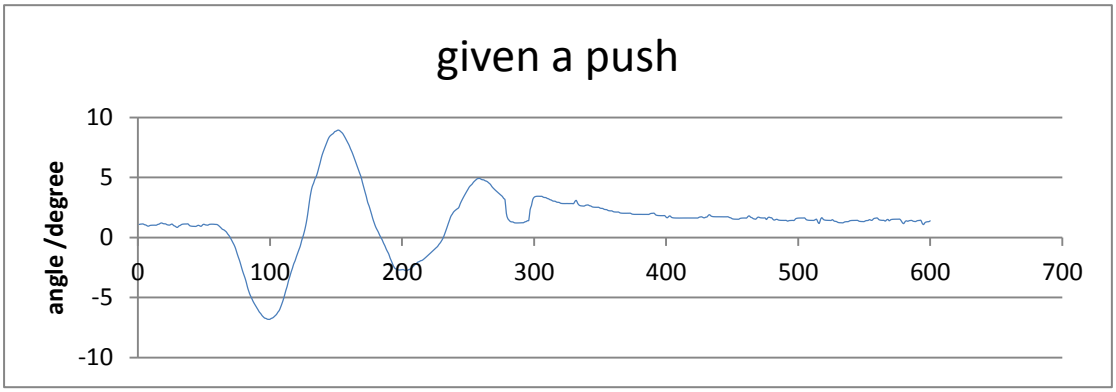

Figure 16 angle distributions while given a push

Figure 17 velocity distributions while given a push

From previous figures we can see that angle sensor is more accurate and generally 1 degree larger than actual value. But for velocity, the value is changing rapidly and quite inaccurate.

7.3.2 Delay of sensors

Despite the noise, the delay of sensors affects the result significantly.

7.4 Result

In a word, the result of implementation is not satisfactory at all. In fact, it would shake more and more severely until it fell down rather than settle down slowly. I believe that more work require to be done before it actually works well.

8 Problems and possible solutions

8.1 Reinforcement learning

The main limitation of reinforcement learning in this project is the discrete value of velocity and angle. In this project, the reinforcement learning we use here is simply to prove the feasibility of this method to solve the problem of push recovery. There are other sophisticated versions of reinforcement learning can be used.

Otherwise I have thought of generating more tables rather than one table, which is kind of similar to associative mapping of paging. For example, we first generate a sparse table like (20, 20), then for the area nearer to axis we generate a more dense table like (40, 40) and so on. In practice, we first look up the velocity and angle in the densest table, if there is not within the range of this table, and then we look up the second densest table which has larger range of values but with sparser grid and so on until we find the action to take in this multi-table policy.

8.2 A star

A star is another way of generating policy and, if better algorithm adopted, the real-time decision of action to take can be achieved. But in this project, it is too slow to be practicable since we have to first compute the expected cost using greedy algorithm for each action.

One solution is first to compute all the possible expected cost before we run the A star algorithm. And then, when running A star algorithm, we may directly look up the expected cost from a file that stores all the cost. I believe it would accelerate the speed of this algorithm a lot, though generating expected cost with discrete value of velocity and angle ahead of time may lead to the same problem in section 1.

8.3 Sensor noise and delay

The key to implement the policy and make sure it works is to match the predictable velocity and angle with actual ones. For motion only, we may compute the average of data from sensor and prediction combined with consideration of noise and delay as the more accurate velocity and angle. Otherwise, some work combined with vision has been done by other teammates.

9 Conclusions

In this project, we mainly observe the feasibility of applying reinforcement learning into push recovery through adjustments of ankles of the robot only. And it turns out that it is reasonable in general. Reinforcement learning is a practicable solution but other algorithm like A star may be an alternative choice. Moreover, we can compare the results from different methods to verify correctness and determine an optimistic solution. There are also many problems mentioned in chapter 8 and more work can be done to improve the efficiency.

Bibliography

[1] http://www.youtube.com/watch?v=3k7rUY5wT7g 2012-11-5.

[2] Hengst Bernhard, Lange Manuel, White Brock. Learning ankle-tilt and foot-placement control for flat-footed bipedal balancing and walking. *2011 11th IEEE-RAS International Conference on Humanoid Robots*, Oct. 2011:288 -293.

[3] Albertus Hendrawan Adiwahono, Chee-Meng Chew, Weiwei Huang, and Van Huan Dau. Humanoid Robot Push Recovery through Walking Phase Modification. *Robotics Automation and Mechatronics (RAM), 2010 IEEE*. June 2010: 569 - 574.

[4] John R. Rebula, Fabian Canas, Jerry E Pratt, Ambarish Goswami. Learning Capture Points for Bipedal Push Recovery. *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference*. May 2008: 1774.

[5] http://en.wikipedia.org/wiki/Reinforcement_learning 2012-11-7

[6] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 1998.

[7] http://en.wikipedia.org/wiki/A*_search_algorithm 2012-11-12

Supplemental Code

A Transition.java

The file implements the discussions of transition fucntion in chapter 5

```
import java.applet.*;
import java.io.*;
import java.util.Scanner;

public class Transition {

    double rad=Math.PI/180;
    double goalang=0;
    double goalvel=0;
    double diffang=rad*2.5;
    double diffvel=rad*2;
    double maxang=rad*17;
    double minang=rad*-13;
    double maxvel=-30*rad;
    double minvel=30*rad;
    double length=0.262;
    double rate=0.99;
    double g=9.8;
    double time=0.01;
    double damp=0.99;
    //new par
    double fwfoot=0.105;
    double bcfoot=-0.057;
    double k=fwfoot/maxang;
    public state start=new state(0*rad,5*rad);
    public state temp=new state(0,0);
    //policy
    int[][]policy;
    //table
    int vel;
    int ang;
    int act;
    //p
    double vincre;
    double aincre;
    //pivot
```

```
double k1=fwfoot/maxang;
double k2=k1/2;
//output
BufferedWriter Out;
BufferedWriter Out2;
BufferedReader in;
//4 test
int counter=0;
//
double[]actions;
//Strut
public class state{
	double velocity;
	double angle;
	public state(double v,double a){
		velocity=v;
		angle=a;
	}
}
//constructor
public Transition(){
}
public double[][][] QueueTable(int vel,int ang,int act,double[]actions) throws IOException{
	 double [][][] Q=new double[vel+1][ang+1][act];
	 policy=new int[vel+1][ang+1];
	 int loops=0;
	 int times=1;
	 this.vel=vel;
	 this.ang=ang;
	 this.act=act;
	 this.actions=actions;
	 vincre=(maxvel-minvel)/vel;
	 aincre=(maxang-minang)/ang;
	 //output file
	 File f=new File("Qv.txt");
	 File f2=new File("p.txt");
	 File f3=new File("E:/A.txt");
	 if(f.exists())f.delete();
	 f.createNewFile();
	 if(f2.exists())f2.delete();
	 f2.createNewFile();
	 System.out.println(f3.getAbsolutePath());
	 FileWriter out=new FileWriter(f);
	 Out=new BufferedWriter(out);
```

```
            FileWriter out2=new FileWriter(f2);
            Out2=new BufferedWriter(out2);
            FileReader In1=new FileReader(f3);
            in=new BufferedReader(In1);
           // */
            System.out.println("loops\ttimes");
            //while(times==1){
            while(times>0){
                  loops++;
                  times=0;
                  for(int i=0;i<vel+1;i++){
                        //System.out.println();
                        //velocity
                        double velocity=minvel+vincre*i;
                        for(int j=0;j<ang+1;j++){
                              //angle
                              double angle=minang+aincre*j;
                              for(int k=0;k<act;k++){
                                    //state now
                                    double action=actions[k];
                                    double reward,maxreward;
                                    double newacc,newvel,newangle,newQ;
                                    int indexvel,indexang;
                                    //the reward

//reward=getReward(velocity,angle,k,actions,Q,vel,ang,vincre,aincre);
                                    //the reward
                                    if(Math.abs(velocity-goalvel)<diffvel                    &&
Math.abs(angle-goalang)<diffang)
                                          reward=1;
                                    else
                                          reward=0;
                                    //next state
                                    state instate=new state(velocity,angle);
                                    state outstate=transition(instate,k,actions);
                                    newvel=outstate.velocity;
                                    newangle=outstate.angle;
                                   // if(newangle>17*rad || newangle<-13)
                                    ///       continue;
                                    // indexvel=(int)((newvel-minvel)/vincre);
                                    //    indexang=(int)((newangle-minang)/aincre);
                                    indexvel=(int)(Math.round((newvel-minvel)/vincre));
                                    indexang=(int)(Math.round((newangle-minang)/aincre));
                                    //test
```

```
                                    //if(indexvel==i && indexang==j)
                                      //System.out.print(i+" "+j+" "+k+"\t");
                                    //

      //System.out.println("a0:"+instate.angle/rad+"\tv0:"+instate.velocity/rad+"\ta1"+newangle/
rad+"\tv1:"+newvel/rad+"("+i+","+j+")    "+indexang+","+indexvel+")");
                                  //System.out.println(("("+angle/rad+","+velocity/rad+")
("+i+","+j+","+k+")                    ("+(newangle-minang)/aincre+","+(newvel-minvel)/vincre+")
("+indexang+","+indexvel+")"));
                                    boolean flag=true;
                                     if(indexvel<0){
                                  //       System.out.println("velocity:          "+newvel/Math.PI*180+"
exceed velocity index: "+indexvel);
                                           indexvel=0;
                                           flag=false;
                                     }
                                     else if(indexvel>=vel+1){
                                  //       System.out.println("velocity:          "+newvel/Math.PI*180+"
exceed velocity index: "+indexvel);
                                           indexvel=vel;
                                           flag=false;
                                     }
                                     if(indexang<0){
                                  //       System.out.println("angle"+newangle/Math.PI*180+"
exceed angle index: "+indexang);
                                           indexang=0;
                                           flag=false;
                                     }
                                     else if(indexang>=ang+1){
                                  //       System.out.println("angle"+newangle/Math.PI*180+"
exceed angle index: "+indexang);
                                           indexang=ang;
                                           flag=false;
                                     }
                                     maxreward=0;
                                    //if(flag && reward==0 && i==indexvel && j==indexang)
                                  //      System.out.println(i+","+j);
                                    /*int count=0;
                                    while(flag && reward==0 && i==indexvel && j==indexang){
                                              // count++;

//System.out.println("("+outstate.angle+","+outstate.velocity+") ");
                                                outstate=transition(outstate,k,actions);
```

```
//System.out.println("("+outstate.angle+","+outstate.velocity+") ");

indexvel=(int)(Math.round((outstate.velocity-minvel)/vincre));

indexang=(int)(Math.round((outstate.angle-minang)/aincre));
                                        //r*=rate;
                                        //r*=0.999;
                                        if(r<0.0001)
                                              break;
                                        if(indexvel<0){
                                                //      System.out.println("velocity:
"+newvel/Math.PI*180+" exceed velocity index: "+indexvel);
                                                        indexvel=0;
                                                        flag=false;
                                                }
                                                else if(indexvel>=vel+1){
                                                //      System.out.println("velocity:
"+newvel/Math.PI*180+" exceed velocity index: "+indexvel);
                                                        indexvel=vel;
                                                        flag=false;
                                                }
                                                if(indexang<0){
                                                //
System.out.println("angle"+newangle/Math.PI*180+" exceed angle index: "+indexang);
                                                        indexang=0;
                                                        flag=false;
                                                }
                                                else if(indexang>=ang+1){
                                                //
System.out.println("angle"+newangle/Math.PI*180+" exceed angle index: "+indexang);
                                                        indexang=ang;
                                                        flag=false;
                                                }
                                        //System.out.println(j+","+i+")
("+indexang+","+indexvel+")");
                                        //System.out.println(r);
                                    }
                                   /*
                                // if(count!=0)
                                if(i==79 && j==44){
                                      System.out.println(velocity/rad+","+angle/rad);
                                      System.out.println(newvel/rad+","+newangle/rad);
                                      System.out.println(k+":"+indexvel+","+indexang);
                                }*/
```

```
                                //          System.out.println(count);
                                for(int n=0;n<act;n++){
                                   if(Q[indexvel][indexang][n]>maxreward)
                                         maxreward=Q[indexvel][indexang][n];
                                }
                                //new value
                                newQ=reward+rate*maxreward;
                                if(newQ!=Q[i][j][k]){
                                      Q[i][j][k]=newQ;
                                      times++;
                                }
                                //System.out.println(k+":"+newQ+"---"+indexvel+","+indexang);
                                //if(i==0 && j==100)
                               //
System.out.println(k+":"+newQ+"---"+indexvel+","+indexang);
                                }
                           }
                     }
                     //get policy
                     int mid=act/2;
                     for(int i=0;i<vel+1;i++){
                          for(int j=0;j<ang+1;j++){
                               boolean flag=true;
                               double r=Q[i][j][mid];
                               policy[i][j]=mid;
                               for(int k=0;k<act;k++){
                                    if(Q[i][j][k]>r){
                                          r=Q[i][j][k];
                                          policy[i][j]=k;
                                    }
                                    if(Q[i][j][k]!=r)
                                          flag=false;

                               }
                               if(flag)
                                    policy[i][j]=-1;
                          }
                     }
                     if(loops%50==1)
                          System.out.println(loops+"\t"+times);
               }
          return Q;
     }
     //the function of transition
```

```
    public state transition(state instate,int act,double[]actions){
        //Acceleration
        double x=pivot(instate.angle,act,actions);
        int positive=x>0?1:-1;
        double l=Math.sqrt(length*length+x*x-2*x*length*Math.cos(Math.PI/2-actions[act]));
        double b=instate.angle-positive*Math.acos((length*length+l*l-x*x)/2/l/length);
        double acc=3.0/4*g*Math.sin(b)/l;
        double v=instate.velocity+acc*time;
        double b2=b+instate.velocity*time+0.5*acc*time*time;
        double a=positive*Math.acos((length*length+l*l-x*x)/2/l/length)+b2;
        //System.out.println("x:"+x+"\tacc:"+acc/rad+"\ta:"+instate.angle/rad+"
\tv:"+instate.velocity/rad+" \t:new \ta "+a/rad+" \tv:"+v/rad);
        //friction
        v*=damp;
        return new state(v,a);
    }
    //another version the function of transition
    public state transition(double v,double a,int act,double[]actions){
        return transition(new state(v,a),act,actions);
    }
    private double pivot(double angle,int act,double[]actions){
        double x=k1*(angle-actions[act])+Math.sin(actions[act])*length;
        if(x<bcfoot)x=bcfoot;
        if(x>fwfoot)x=fwfoot;
        return x;
    }
    public double getReward(double velocity,double angle,int
act,double[]actions,double[][][]Q,int vel,int ang, double vincre,double aincre){
        //state now
          double action=actions[act];
          double reward,maxreward;
          double newacc,newvel,newangle,newQ;
          int indexvel,indexang;
        //the reward
          if(Math.abs(velocity-goalvel)<diffvel && Math.abs(angle-goalang)<diffang)
              reward=1;
          else
              reward=0;
          //next state
          newangle=angle+action;
          newacc=3*g*Math.sin(newangle)/length;
          newvel=velocity+time*newacc;
          newangle=newangle+newvel*time;  //+.5*newacc*time*time;
          indexvel=(int)((newvel-minvel)/vincre);
```

```
            indexang=(int)((newangle-minang)/aincre);
            if(indexvel<0){
        //      System.out.println("velocity: "+newvel/Math.PI*180+" exceed velocity index: "+indexvel);
                indexvel=0;
            }
            else if(indexvel>=vel){
        //      System.out.println("velocity: "+newvel/Math.PI*180+" exceed velocity index: "+indexvel);
                indexvel=vel-1;
            }
            if(indexang<0){
        //      System.out.println("angle"+newangle/Math.PI*180+" exceed angle index: "+indexang);
                indexang=0;
            }
            else if(indexang>=ang){
        //      System.out.println("angle"+newangle/Math.PI*180+" exceed angle index: "+indexang);
                indexang=ang-1;
            }
            maxreward=0;
            for(int n=0;n<act;n++){
                if(Q[indexvel][indexang][n]>maxreward)
                    maxreward=Q[indexvel][indexang][n];
            }
            return 1;
    }
    public int[][]getPolicy(){
        return policy;
    }
    public void print(int x,int y,int z,double[][][]Q) throws IOException{
        System.out.print("\t");
        Out.write("\t");
        for(int j=0;j<y+1;j++){
            System.out.print(((maxang-minang)/y*j+minang)/rad+"\t");
            Out.write(((maxang-minang)/y*j+minang)/rad+"\t");
        }
        Out.write("\n");
        System.out.println();
        for(int i=0;i<x+1;i++){
            System.out.print(((maxvel-minvel)/x*i+minvel)/rad+"\t");
            Out.write(((maxvel-minvel)/x*i+minvel)/rad+"\t");
            for(int j=0;j<y+1;j++){
```

```
                System.out.print("{");
                Out.write("{");
                double r=0;
                for(int k=0;k<z;k++){
                    //System.out.print(Q[i][j][k]+",");
                    if(Q[i][j][k]>r)
                        r=Q[i][j][k];
                    Out.write(Q[i][j][k]+",");
                }
                System.out.print(r);
                System.out.print("} \t");
                Out.write("} \t");
            }
            System.out.println();
            Out.write("\n");
        }
        Out.close();
    }
    public void print(int x,int y,int[][]P) throws IOException{
        System.out.print("\t");
        System.out.print(minvel+"\t"+maxvel);
        System.out.println(minang+"\t"+maxang);
        System.out.println(vel+"\t"+ang);
        Out2.write(act+"\t");
        for(int i=0;i<act;i++)
            Out2.write(actions[i]+"\t");
        Out2.write(minvel+"\t"+maxvel);
        Out2.write("\t");
        Out2.write(minang+"\t"+maxang);
        Out2.write("\t");
        Out2.write((vel)+"\t"+(ang));
        Out2.write("\t");
        for(int i=0;i<x+1;i++){
                for(int j=0;j<y+1;j++){
                    System.out.print(P[i][j]+"\t");
                    Out2.write(P[i][j]+"\t");
                }
                System.out.println();
                Out2.write("\n");
            }
        Out2.close();
    }
    public void calAcc() throws IOException{
        File f3=new File("A.txt");
```

```
            FileReader In1=new FileReader(f3);
            in=new BufferedReader(In1);
            double[]d={0};
            this.actions=d;
           while(in.ready()){
                String S=in.readLine();
                double anlge=(Double.parseDouble(S)+0.9970536)*rad;
                double x=pivot(anlge,0,actions);
                int positive=x>0?1:-1;
                double
l=Math.sqrt(length*length+x*x-2*x*length*Math.cos(Math.PI/2-actions[act]));
                double b=anlge-positive*Math.acos((length*length+l*l-x*x)/2/l/length);
                double acc=3.0/4*g*Math.sin(b)/l;
                System.out.println(-acc/rad);
           }
      }
}
```

B Qtable.java

This file implements the Q table discussed in the chapter 5

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

import javax.swing.*;
import java.math.*;

public class Qtable {

      double rad=Math.PI/180;
      double goalang=0;
      double goalvel=0;
      double diffang=rad*3;
      double diffvel=rad*10;
      double maxang=rad*17;
      double minang=rad*-13;
      double maxvel=Math.sqrt(2-2*Math.cos(maxang));
      double minvel=-maxvel;
      double length=0.573;
```

```
double rate=0.99;
double g=9.8;
double time=0.01;

public double[][][] QueueTable(int vel,int ang,int act,double[]actions) throws IOException{
      double [][][] Q=new double[vel][ang][act];
       int loops=0;
       int times=1;
       double vincre=(maxvel-minvel)/vel;
       double aincre=(maxang-minang)/ang;
       //output file
       File f=new File("data.txt");
       if(f.exists())f.delete();
       f.createNewFile();
       FileWriter out=new FileWriter(f);
       BufferedWriter Out=new BufferedWriter(out);
       //
       System.out.println("loops\ttimes");
       while(times>0){
            loops++;
            times=0;
            for(int i=0;i<vel;i++){
                 //velocity
                 double velocity=minvel+vincre*i;
                 for(int j=0;j<ang;j++){
                      //angle
                      double angle=minang+aincre*j;
                      for(int k=0;k<act;k++){
                           //state now
                           double action=actions[k];
                           double reward,maxreward;
                           double newacc,newvel,newangle,newQ;
                           int indexvel,indexang;
                           //the reward

//reward=getReward(velocity,angle,k,actions,Q,vel,ang,vincre,aincre);
                          //the reward
                           if(Math.abs(velocity-goalvel)<diffvel                    &&
Math.abs(angle-goalang)<diffang)
                                reward=1;
                           else
                                reward=0;
                           //next state
                           newangle=angle+action;
```

```
                                newacc=3*g*Math.sin(newangle)/length;
                                newvel=velocity+time*newacc;
                                newangle=newangle+newvel*time;  //+.5*newacc*time*time;
                                indexvel=(int)((newvel-minvel)/vincre);
                                indexang=(int)((newangle-minang)/aincre);
                                if(indexvel<0){
                            //      System.out.println("velocity:          "+newvel/Math.PI*180+"
exceed velocity index: "+indexvel);
                                    indexvel=0;
                                }
                                else if(indexvel>=vel){
                            //      System.out.println("velocity:          "+newvel/Math.PI*180+"
exceed velocity index: "+indexvel);
                                    indexvel=vel-1;
                                }
                                if(indexang<0){
                            //      System.out.println("angle"+newangle/Math.PI*180+"
exceed angle index: "+indexang);
                                    indexang=0;
                                }
                                else if(indexang>=ang){
                            //      System.out.println("angle"+newangle/Math.PI*180+"
exceed angle index: "+indexang);
                                    indexang=ang-1;
                                }
                                maxreward=0;
                                for(int n=0;n<act;n++){
                                    if(Q[indexvel][indexang][n]>maxreward)
                                        maxreward=Q[indexvel][indexang][n];
                                }
                                //new value
                                newQ=reward+rate*maxreward;
                                if(newQ!=Q[i][j][k]){
                                    Q[i][j][k]=newQ;
                                    times++;
                                }
                            }
                        }
                    }
                    System.out.println(loops+"\t"+times);
                }

            return Q;
        }
```

```
    public double getReward(double velocity,double angle,int act,double[]actions,double[][][]Q,int vel,int ang, double vincre,double aincre){
        //state now
          double action=actions[act];
          double reward,maxreward;
          double newacc,newvel,newangle,newQ;
          int indexvel,indexang;
        //the reward
          if(Math.abs(velocity-goalvel)<diffvel && Math.abs(angle-goalang)<diffang)
              reward=1;
          else
              reward=0;
          //next state
          newangle=angle+action;
          newacc=3*g*Math.sin(newangle)/length;
          newvel=velocity+time*newacc;
          newangle=newangle+newvel*time;  //+.5*newacc*time*time;
          indexvel=(int)((newvel-minvel)/vincre);
          indexang=(int)((newangle-minang)/aincre);
          if(indexvel<0){
        //      System.out.println("velocity: "+newvel/Math.PI*180+" exceed velocity index: "+indexvel);
              indexvel=0;
          }
          else if(indexvel>=vel){
        //      System.out.println("velocity: "+newvel/Math.PI*180+" exceed velocity index: "+indexvel);
              indexvel=vel-1;
          }
          if(indexang<0){
        //      System.out.println("angle"+newangle/Math.PI*180+" exceed angle index: "+indexang);
              indexang=0;
          }
          else if(indexang>=ang){
        //      System.out.println("angle"+newangle/Math.PI*180+" exceed angle index: "+indexang);
              indexang=ang-1;
          }
          maxreward=0;
          for(int n=0;n<act;n++){
              if(Q[indexvel][indexang][n]>maxreward)
                  maxreward=Q[indexvel][indexang][n];
          }
```

```
            return 1;

        }
}
```

C Plot.java and Arrow.java

This file generates all kinds of plots to test especially shown in the chapter 6

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.lang.System;
import javax.swing.*;

public class plot extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    Transition T;
    Astar A;
    recursive R;
    Transition.state start;
    int act;
    double[]actions;
    double adiff=0.001;
    double vdiff=0.001;
    int height=900;
    int width=1200;
    int x0=600;
    int y0=400;
    int count=0;
    double length=15;
    int[][]policy;
    int vel;
    int ang;
    double rad=Math.PI/180;
    double goalang=0;
    double goalvel=0;
    double diffang=rad*2;
```

```
double diffvel=rad*2;
double maxang;;
double minang;
int last=1;
boolean flag=true;
public plot(Transition T,Transition.state start,int act,double[]actions,int[][]policy)
{
	super("plot");
	this.T=T;
	this.start=start;
	this.A=new Astar(actions);
	this.act=act;
	this.actions=actions;
	this.policy=policy;
	this.maxang=T.maxang;
	this.minang=T.minang;
	this.vel=T.vel;
	this.ang=T.ang;
	setBounds(0,0,1200,1000);
	setVisible(true);
}
@SuppressWarnings("deprecation")
public void paint(Graphics g)
{
	//draw info
	g.drawString("vel: "+vel+"    "+"ang: "+ang, 10, 40);
	String s="actions :     {";
	for(int i=0;i<actions.length;i++)
		s+=(actions[i]/T.rad+" ");
	s+="}";
	g.drawString(s, 10, 60);
	g.drawString("Time step :     "+T.time, 10, 80);
	g.drawString("Damp :     "+T.damp, 10, 100);
	Date d=new Date();
	g.drawString(d.toLocaleString(), 10, 120);
	//
	g.drawLine(x0, 0, x0, height);
	g.drawLine(0,y0, width, y0);
	g.drawString("velocity\nrad/s", x0+20, 40);
	g.drawString("angle\nrad", 1000, y0-20);
	for(int i=-20;i<=20;i++){
		T.temp.angle=i*T.rad;
		g.drawLine(getX(T.temp), y0-5,getX(T.temp),y0+5);
		g.drawString(i+"", getX(T.temp), y0+15);
```

```
        }
        T.temp.angle=0;
        for(int i=-30;i<=30;i++){
            T.temp.velocity=2*i*T.rad;
            g.drawLine(x0-5, getY(T.temp),x0+5,getY(T.temp));
            g.drawString(2*i+"", x0-35, getY(T.temp));
        }
        T.temp.velocity=0;
        draw(start,0,g);
        /*********************************************/
        start.angle=      5*T.rad;
        start.velocity=   0*T.rad;
        g.drawString("start point: ("+start.angle/T.rad+","+start.velocity/T.rad+")", 10, 150);
        //R=new recursive(start.velocity,start.angle,actions);
        /*********************************************/
        //next(start,g,0);
        count=0;
        last=1;
        next(start,g,1);
        System.out.println(flag);
        flag=false;
        //next(start,g,2);
        /*
        for(int i=-5;i<=5;i++)
        {
            g.setColor(Color.YELLOW);
            draw(start,4,g);
            start.angle=i*T.rad;
            next(start,g,0);
            next(start,g,1);
            next(start,g,2);
        }*/
        g.drawString("Total steps: "+count, 10, 170);
        System.out.println("steps: "+count);
    }
    private void draw(Transition.state S,int i,Graphics g){
        Color color=Color.YELLOW;
        switch(i)
        {
        case 0:color=Color.RED;break;
        case 1:color=Color.GREEN;break;
        case 2:color=Color.BLUE;break;
        }
        g.setColor(color);
```

```
            g.fillOval(getX(S)-2,getY(S)-2,4,4);
        }
        private void next(Transition.state S,Graphics g){
            for(int i=0;i<act;i++){
                Transition.state N=T.transition(S, i, actions);
                //System.out.println(Math.abs(N.angle-S.angle)+"
"+Math.abs(N.velocity-S.velocity));
                //if(N.angle>T.maxang || N.angle<T.minang || N.velocity>T.maxvel||
N.velocity<T.minvel)
                    //continue;
                if(Math.abs(N.angle-S.angle)>adiff && Math.abs(N.velocity-S.velocity)>vdiff){
                    draw(N,i,g);
                    next(N,g);
                }
            }
        }
        private void next(Transition.state S,Graphics g,int act){
            int p=0;
            //--RL--
            //p=RL(S);
            //--greedy--
            //p=greedy(S);
            //--Recursive--
            //R.computevalue();
            //--Astar--
            //p=A.findAction(S);
            //System.out.println(A.findActions(S));
            String[]ps=A.findActions(S).split(" ");
            if(ps.length>0){
                count=ps.length-1;
                for(int i=0;i<ps.length;i++){
                    p=Integer.parseInt(ps[i]);
                    Transition.state N=T.transition(S,p , actions);
                    draw(N,p,g);
                    S=N;
                }
            }
            //System.out.println(count+" "+p+"\t"+S.velocity/T.rad+","+S.angle/T.rad);

        //System.out.println(S.angle/T.rad+":"+(T.maxang-T.minang)+":"+ang+":"+(S.angle-T.minang)
);
            //System.out.println(iv+","+ia+":\t"+p);
            //int p=0;
            if(p==-1)
```

```
                p=last;
            else
                last=p;
            Transition.state N=T.transition(S,p , actions);
            //System.out.println(iv+","+ia+":"+p);
            if(N.angle>T.maxang || N.angle<T.minang)
                return;
            if(Math.abs(N.angle)<T.diffang && Math.abs(N.velocity)<T.diffvel){
                draw(N,p,g);
                return;
            }
            //

        //System.out.println(S.angle/rad+"\t"+Math.ceil((T.maxang-T.minang)/Math.abs((N.angle-S.
angle)*2))+"\t"+S.velocity/rad+"\t"+Math.ceil((T.maxang-T.minang)/Math.abs((N.velocity-S.veloci
ty)/2)));
            //
            //System.out.println((Math.abs(N.angle)>adiff || Math.abs(N.velocity)>vdiff));
            draw(N,p,g);
            //int indexvel=(int)( Math.round((N.velocity-T.minvel)/((T.maxvel-T.minvel)/50)));
            //int indexang=(int)(Math.round((N.angle-T.minang)/((T.maxang-T.minang)/50)));
            //System.out.println(count+": "+indexang+" "+indexvel);
            count++;
            //System.out.println(count);
            next(N,g,act);
            //System.out.println(count);
        }
        private int getX(Transition.state S){
            return (int) (500/(T.maxang)*S.angle)+x0;
        }
        private int getY(Transition.state S){
            return (int)(200/(T.maxvel)*S.velocity)+y0;
        }
        private void drawarrow(double x1,double y1,double x2,double y2,Graphics g){
            double distance=Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
            double x3=x2-(x2-x1)/distance*length;
            double y3=y2-(y2-y1)/distance*length;
            double angle=angle(x1,y1,x2,y2);
            double l=length*Math.sqrt(3)/2;
            double a=l*Math.sin(angle);
            double b=l*Math.cos(angle);
            g.drawLine((int)x2,(int)y2,(int)(x3+a), (int)(y3-b));
            g.drawLine((int)x2,(int)y2, (int)(x3-a),(int)(y3+b));
        }
```

```
 private double angle(double x1,double y1,double x2,double y2){
        if(x1==x2)
        {
            if(y2>y1)
                return Math.PI/2;
            else
                return Math.PI*3/2;
        }
        else
            return x2>x1?Math.atan((y2-y1)/(x2-x1)):Math.atan((y2-y1)/(x2-x1))+Math.PI;
    }

    private int RL(Transition.state S){
        int iv=(int)( Math.round((S.velocity-T.minvel)/((T.maxvel-T.minvel)/vel)));
        int ia=(int)(Math.round((S.angle-T.minang)/((T.maxang-T.minang)/ang)));
        if(iv>vel)
            iv=vel;
        if(ia>ang)
            ia=ang;
        if(iv<0)
            iv=0;
        if(ia<0)
            ia=0;
        int p=policy[iv][ia];
        return p;
    }

    private int greedy(Transition.state S){
        double distance=Double.MAX_VALUE;
        int p=1;
        for(int i=0;i<actions.length;i++){
            Transition.state N=T.transition(S,i, actions);
            double d=N.angle*N.angle+N.velocity*N.velocity;
            if(d<distance){
                distance=d;
                p=i;
            }
        }
        return p;
    }

}

//same as arrow.java
```

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.lang.System;
import javax.swing.*;

public class arrow extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    Transition T;
    Transition.state start;
    int act;
    double[]actions;
    double adiff=0.000001;
    double vdiff=0.000001;
    int height=900;
    int width=1200;
    int x0=600;
    int y0=400;
    int count=0;
    int vel;
    int ang;
    double rad=Math.PI/180;
    double goalang=0;
    double goalvel=0;
    double diffang=rad*2;
    double diffvel=rad*2;
    double maxang=rad*30;
    double minang=rad*-20;
    //double maxvel=-Math.sqrt(2-2*Math.cos(maxang));
    //double minvel=Math.sqrt(2-2*Math.cos(minang));
    double maxvel=-50*rad;
    double minvel=50*rad;
    double vincre;
    double aincre;
    double length=15;
    double scalex=350;
    double scaley=350;
```

```
public arrow(int vel,int ang,Transition T,Transition.state start,int act,double[]actions)
{
	super("arrow");
	this.T=T;
	this.start=start;
	this.act=act;
	this.actions=actions;
	this.vel=vel;
	this.ang=ang;
	vincre=(maxvel-minvel)/vel;
	aincre=(maxang-minang)/ang;
	setBounds(0,0,1200,1000);
	setVisible(true);
}
public void paint(Graphics g)
{
	g.drawLine(x0, 0, x0, height);
	g.drawLine(0,y0, width, y0);
	g.drawString("velocity\nrad/s", x0+20, 40);
	g.drawString("angle\nrad", 1000, y0-20);
	for(int i=-20;i<=30;i++){
		T.temp.angle=i*T.rad;
		g.drawLine(getX(T.temp), y0-5,getX(T.temp),y0+5);
		g.drawString(i+"", getX(T.temp), y0+15);
	}
	T.temp.angle=0;
	for(int i=-20;i<=20;i++){
		T.temp.velocity=2*i*T.rad;
		g.drawLine(x0-5, getY(T.temp),x0+5,getY(T.temp));
		g.drawString(2*i+"", x0-35, getY(T.temp));
	}
	for(int i=0;i<vel+1;i++){
		start.velocity=minvel+i*vincre;
		start.angle=0;
		//int s=vel/2;
		//i+=Math.abs(s-i)/(vel/40);
		int y=getY(start);
		g.drawLine(0,y,width,y);
		g.drawString(i+"", 20, y+20);
	}
	for(int i=0;i<ang+1;i++){
		start.angle=minang+i*aincre;
		start.velocity=0;
```

```
            //int s=ang/2;
            //i+=Math.abs(s-i)/(vel/40);
            int y=getY(start);
            int x=getX(start);
            g.drawLine(x, 0, x, height);
            g.drawString(i+"", x+20, 50);
        }
        T.temp.velocity=0;
        /*****************************************/
        start.angle=    0.2*rad;
        start.velocity=-20 *rad;
        /*****************************************/
        next(start,g,0,0);
        //next2(start,g,1,0);
        //next2(start,g,2,0);
        /*g.setColor(Color.YELLOW);
        //draw(start, start,4,g);
        for(int i=15;i<16;i++)
        {
            start.angle=i*rad;
            start.velocity=0;
            //draw(start,start,0,g);
            next(start,g,0,0);
            next(start,g,1,0);
            next(start,g,2,0);
        }

        for(int i=0;i<vel+1;i++){
            for(int j=0;j<ang+1;j++){
                for(int k=1;k<2;k++){
                    start.velocity=minvel+vincre*i;
                    start.angle=minang+aincre*j;
                    Transition.state N=T.transition(start,k , actions);
                    int indexang=(int)(Math.round((N.angle-minang)/aincre));
                    int indexvel=(int)(Math.round((N.velocity-minvel)/vincre));
                    N.angle=getA(indexang);
                    N.velocity=getV(indexvel);
                    draw(start,N,k,g);
                }
            }
        }*/
    }
    private void draw(Transition.state O, Transition.state S,int i,Graphics g){
        Color color=Color.YELLOW;
```

```
        switch(i)
        {
        case 0:color=Color.RED;break;
        case 1:color=Color.GREEN;break;
        case 2:color=Color.BLUE;break;
        }
        g.setColor(color);
        int x1=getX(O);
        int y1=getY(O);
        int x2=getX(S);
        int y2=getY(S);
        if(x1==x2 && y1==y2)
                return;
        /*int x1=getx(O);
        int y1=gety(O);
        int x2=getx(S);
        int y2=gety(S);*/
        //System.out.println(x1+" "+y1);
        g.fillOval(x1,y1,5,5);
        g.drawLine(x1 ,y1 ,x2,y2);
        drawarrow(x1,y1,x2,y2,g);
        /*double a=y2>y1?90*rad:-90*rad;
        if(x1!=x2)
                a=Math.atan((y1-y2)/(x1-x2));
        g.drawLine(x2, y2, x2+2, (int)Math.tan(a+30*rad)*x2+y2);
        g.drawLine(x2, y2, x2-2, (int)Math.tan(a-30*rad)*x2+y2);
        */
}
private void next2(Transition.state S,Graphics g,int act,int count){
        if(Math.abs(S.angle)<diffang && Math.abs(S.velocity)<diffvel)
                return;
        Transition.state N=T.transition(S, act, actions);
        int ia=(int)(Math.round((S.angle-minang)/aincre));
        int iv=(int)(Math.round((S.velocity-minvel)/vincre));
        int indexang=(int)(Math.round((N.angle-minang)/aincre));
        int indexvel=(int)(Math.round((N.velocity-minvel)/vincre));
        System.out.println("("+ia+","+iv+") ("+indexang+","+indexvel+")");
        if(ia==indexang && iv==indexvel){
                for(int i=0;i<3;i++){
                        int iang=indexang;
                        int ivel=indexvel;
                        while(ia==iang && iv==ivel){
                                System.out.print("("+N.angle+","+N.angle+") ");
                                N=T.transition(N, i, actions);
```

```
                    System.out.println("("+N.angle+","+N.angle+") ");
                    iang=(int)(Math.round((N.angle-minang)/aincre));
                    ivel=(int)(Math.round((N.velocity-minvel)/vincre));
                }
                System.out.println(i);
                N.angle=getA(iang);
                N.velocity=getV(ivel);
                draw(S,N,i,g);
                next2(N,g,act,0);
            }
        }
        else if(Math.abs(N.angle)>adiff || Math.abs(N.velocity)>vdiff){
            N.angle=getA(indexang);
            N.velocity=getV(indexvel);
            draw(S,N,act,g);
            System.out.println("("+S.angle+","+S.velocity+") ");
            System.out.println("("+N.angle+","+N.velocity+") ");
            next2(N,g,act,0);
        }
    }
    private void next1(Transition.state S,Graphics g,int act,int count){
        if(Math.abs(S.angle)<diffang && Math.abs(S.velocity)<diffvel)
            return;
        Transition.state N=T.transition(S, act, actions);
        //System.out.println(S.angle/rad+"      "+S.velocity/rad+"      ----      "+N.angle/rad+"
"+N.velocity/rad);
        int indexang=(int)(Math.round((N.angle-minang)/aincre));
        int indexvel=(int)(Math.round((N.velocity-minvel)/vincre));
        N.angle=getA(indexang);
        N.velocity=getV(indexvel);
        if(count>2)
            return;
        if(N.angle>T.maxang || N.angle<T.minang)
            return;
        if(Math.abs(N.angle-S.angle)<adiff && Math.abs(N.velocity-S.velocity)<vdiff){
            for(int i=0;i<this.act;i++){
                if(i!=act){
                    next1(N,g,i,count+1);
                }
            }
            return;
        }
        if(Math.abs(N.angle-actions[act])>adiff || Math.abs(N.velocity)>vdiff){
            draw(S,N,act,g);
```

```
                next1(N,g,act,0);
            }
            //System.out.println(count);
        }
        private void next(Transition.state S,Graphics g,int act,int count){
            if(Math.abs(S.angle)<diffang && Math.abs(S.velocity)<diffvel)
                return;
            Transition.state N=T.transition(S, act, actions);
            //System.out.println(S.angle/rad+"        "+S.velocity/rad+"        ----        "+N.angle/rad+"
"+N.velocity/rad);
            int indexang=(int)(Math.round((N.angle-minang)/aincre));
            int indexvel=(int)(Math.round((N.velocity-minvel)/vincre));
            N.angle=getA(indexang);
            N.velocity=getV(indexvel);
            if(N.angle>T.maxang || N.angle<T.minang)
                return;
            if(Math.abs(N.angle-S.angle)<adiff && Math.abs(N.velocity-S.velocity)<vdiff)
                return;
            if(Math.abs(N.angle)>adiff || Math.abs(N.velocity)>vdiff){
                draw(S,N,act,g);
                next(N,g,act,0);
            }
            //System.out.println(count);
        }
        private int getX(Transition.state S){
            return (int) (scalex/(T.maxang)*S.angle)+x0;
        }
        private int getY(Transition.state S){
            return (int)(scaley/(T.maxvel)*S.velocity)+y0;
        }
        private int getx(Transition.state S){
            int indexang=(int)(Math.round(S.angle/aincre));
            double a=indexang*aincre+T.minang;
            return (int) (scalex/(T.maxang)*a)+x0;
        }
        private int gety(Transition.state S){
            int indexvel=(int)(Math.round((S.velocity)/vincre));
            double v=indexvel*vincre+T.minvel;
            return (int)(scaley/(T.maxvel)*v)+y0;
        }
        private double getA(int index){
            return minang+index*aincre;
        }
        private double getV(int index){
```

```
        return minvel+index*vincre;
    }
    private void drawarrow(double x1,double y1,double x2,double y2,Graphics g){
        double distance=Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
        double x3=x2-(x2-x1)/distance*length;
        double y3=y2-(y2-y1)/distance*length;
        double angle=angle(x1,y1,x2,y2);
        double l=length*Math.sqrt(3)/2;
        double a=l*Math.sin(angle);
        double b=l*Math.cos(angle);
        g.drawLine((int)x2,(int)y2,(int)(x3+a), (int)(y3-b));
        g.drawLine((int)x2,(int)y2, (int)(x3-a),(int)(y3+b));
    }
    private double angle(double x1,double y1,double x2,double y2){
        if(x1==x2)
        {
            if(y2>y1)
                return Math.PI/2;
            else
                return Math.PI*3/2;
        }
        else
            return x2>x1?Math.atan((y2-y1)/(x2-x1)):Math.atan((y2-y1)/(x2-x1))+Math.PI;
    }
}
```

D A star.java

This file implements the A star algorithm discussed in the chapter 7.

```
import java.applet.*;
import java.io.*;
import java.util.*;

public class Astar {
    Transition T=new Transition();
    //LinkedList<state> path=new LinkedList<state>();
    PriorityQueue<state> path=new PriorityQueue<state> ();
    double[]actions;
    int c=0;
    int step;

    public Astar(double[]actions){
        this.actions=actions;
```

```
}
public class state implements Comparable<state>{
	public double velocity;
	public double angle;
	public int step;
	public int expect;
	public int act;
	public String acts;
	public state(){

	}
	public state(double v,double a,int s,int act,double[]actions){
		this.velocity=v;
		this.angle=a;
		this.step=s;
		this.act=act;
		expect=greedyexp(v,a);
	}
	public state(double v,double a,int s,int act,String acts,double[]actions){
		this.velocity=v;
		this.angle=a;
		this.step=s;
		this.act=act;
		this.acts=acts;
		expect=greedyexp(v,a);
	}
	public state(Transition.state S,int s,int act,double[]actions){
		this.velocity=S.velocity;
		this.angle=S.angle;
		this.step=s;
		this.act=act;
		expect=greedyexp(velocity,angle);
	}
	public state(Transition.state S,int s,int act,String acts,double[]actions){
		this.velocity=S.velocity;
		this.angle=S.angle;
		this.step=s;
		this.act=act;
		this.acts=acts;
		expect=greedyexp(velocity,angle);
	}
	//public operator
	@Override
	public int compareTo(state other) {
```

```
            if(this.step+this.expect>other.step+other.expect)
                return 1;
            else if(this.step+this.expect<other.step+other.expect)
                return -1;
            return 0;
        }
    }
    public int findAction(double v,double a){
        boolean flag=true;
        int policy=0;
        c=0;
        path.clear();
        for(int i=0;i<actions.length;i++){
            path.add(new state(T.transition(v, a, i,actions),0,i,actions));
        }
        while(flag){
            c++;
            //System.out.println(path.peek().velocity/T.rad+","+path.peek().angle/T.rad);
            state shortest=path.poll();
            for(int i=0;i<actions.length;i++){
                Transition.state N=T.transition(shortest.velocity, shortest.angle,i, actions);
                path.add(new state(N.velocity,N.angle,shortest.step+1,shortest.act,actions));
                if((Math.abs(N.velocity-T.goalvel)<T.diffvel                    &&
Math.abs(N.angle-T.goalang)<T.diffang)){
                    this.step=shortest.step+1;
                    flag=false;
                    policy=shortest.act;
                    break;
                }
            }
        }
        System.out.println(c);
        return policy;
    }
    public String findActions(double v,double a){
        boolean flag=true;
        int policy=0;
        c=0;
        String acts="";
        path.clear();
        for(int i=0;i<actions.length;i++){
            path.add(new state(T.transition(v, a, i,actions),0,i,i+"",actions));
        }
        while(flag){
```

```
                c++;
                //System.out.println(path.peek().velocity/T.rad+","+path.peek().angle/T.rad);
                state shortest=path.poll();
                for(int i=0;i<actions.length;i++){
                    Transition.state N=T.transition(shortest.velocity, shortest.angle,i, actions);
                    path.add(new
state(N.velocity,N.angle,shortest.step+1,shortest.act,shortest.acts+" "+i,actions));
                    if((Math.abs(N.velocity-T.goalvel)<T.diffvel                    &&
Math.abs(N.angle-T.goalang)<T.diffang)){
                        this.step=shortest.step+1;
                        flag=false;
                        policy=shortest.act;
                        acts=shortest.acts+" "+i;
                        break;
                    }
                }
            }
            System.out.println(c);
            return acts;
        }
        public int findAction(Transition.state S){
            return findAction(S.velocity,S.angle);
        }
        public String findActions(Transition.state S){
            return findActions(S.velocity,S.angle);
        }
        public int expectCost(double v,double a){
            int t1,t2;
            int sum1,sum2;
            t1=t2=0;
            double vel,ang;
            vel=v;
            ang=a;
            while(Math.abs(ang-T.goalang)>T.diffang){
                t1++;
                T.start=T.transition(vel, ang, 0, actions);
                vel=T.start.velocity;
                ang=T.start.angle;
                if(ang>T.maxang || ang<T.minang){
                    t1=Integer.MAX_VALUE;
                    break;
                }
            }
            while(Math.abs(vel-T.goalvel)>T.diffvel){
```

```
            t2++;
            T.start=T.transition(vel, ang, actions.length-1, actions);
            vel=T.start.velocity;
            ang=T.start.angle;
            if(vel<T.maxvel || vel>T.minvel){
                t2=Integer.MAX_VALUE;
                break;
            }
        }
        sum1=t1<t2?t2:t1;
        vel=v;
        ang=a;
        t1=t2=0;
        while(Math.abs(ang-T.goalang)>T.diffang){
            t1++;
            T.start=T.transition(vel, ang, actions.length-1, actions);
            vel=T.start.velocity;
            ang=T.start.angle;
            if(ang>T.maxang || ang<T.minang){
                t1=Integer.MAX_VALUE;
                break;
            }
        }
        while(Math.abs(vel-T.goalvel)>T.diffvel){
            t2++;
            T.start=T.transition(vel, ang, 0, actions);
            vel=T.start.velocity;
            ang=T.start.angle;
            if(vel<T.maxvel || vel>T.minvel){
                t1=Integer.MAX_VALUE;
                break;
            }
        }
        sum2=t1<t2?t2:t1;
        //System.out.println(v/T.rad+"\t"+a/T.rad+"\t"+t1+"\t"+t2);
        return sum1>sum2?sum2:sum1;
    }
    public int expectCost0(double v,double a){

        return 0;

    }
    public int greedyexp(double velocity,double angle){
        int count=0;
```

```
            while(Math.abs(velocity)>T.diffvel|| Math.abs(angle)>T.diffang){
                if(angle>T.maxang || angle<T.minang){
                    return Integer.MAX_VALUE;
                }
                double nextv=0;
                double nexta=0;
                count++;
                double distance=Double.MAX_VALUE;
                for(int i=0;i<actions.length;i++){
                    Transition.state N=T.transition(velocity,angle,i, actions);
                    double d=0;
                    //d=(N.angle)*(N.angle)+(N.velocity)*(N.velocity);
                    if(N.angle>0 && N.velocity>0)
                        d=(N.angle-2.5)*(N.angle-2.5)+(N.velocity-2)*(N.velocity-2);
                    else if(N.angle<0 && N.velocity>0)
                        d=(N.angle+2.5)*(N.angle+2.5)+(N.velocity-2)*(N.velocity-2);
                    else if(N.angle>0 && N.velocity<0)
                        d=(N.angle-2.5)*(N.angle-2.5)+(N.velocity+2)*(N.velocity+2);
                    else
                        d=(N.angle+2.5)*(N.angle+2.5)+(N.velocity+2)*(N.velocity+2);/**/
                    if(d<distance){
                        distance=d;
                        nextv=N.velocity;
                        nexta=N.angle;
                    }
                }
                velocity=nextv;
                angle=nexta;
            }
            return count;
        }
}
```

E WalkEngine.cpp

Partial code in WalkEngine.cpp that implement the policy discussed in the chapter 7

```
    j.angles[Joints::LAnklePitch]    = -ApL + request.jointOffset[L][PITCH][ANKLE];
    j.angles[Joints::RAnklePitch]    = -ApR + request.jointOffset[R][PITCH][ANKLE];
    //initial value
    float initial=DEG2RAD(-25);
    //in,output
    if(policy==NULL)
```

```
    {
        ifstream in("/home/nao/data/p.txt");
        if(in==NULL)
        {
            cout<<"file doesn't exit!"<<endl;
        }
        else
        {
            cout<<"file loaded!"<<endl;
            in>>act;
            cout<<act<<" actions: ";
            actions=new double[act];
            for(int i=0;i<act;i++)
            {
                in>>actions[i];
                cout<<actions[i]<<" ";
            }
            cout<<endl;
            in>>minvel>>maxvel>>minang>>maxang;
            in>>vindex>>aindex;
            cout<<minvel<<" "<<maxvel<<" "<<minang<<" "<<maxang<<endl;
            cout<<vindex<<" "<<aindex<<endl;
            policy=new int[(vindex+1)*(aindex+1)];
            int index;
            for(int i=0;i<vindex+1;i++)
            {
                for(int j=0;j<aindex+1;j++)
                {
                    index=(vindex+1)*i+j;
                    in>>policy[index];
                }
            }
        }
    }
    cout<<"
"<<count<<endl;
    double velocity=    sensors.sensors[Sensors::InertialSensor_GyrY];
    double angle=   sensors.sensors[Sensors::InertialSensor_AngleY];
    if(++count>80)
    {
    count=0;
    int ivel=       getindex(minvel,maxvel,vindex,velocity);
    int iang=       getindex(minang,maxang,aindex,angle);
    int p=policy[ivel*(vindex+1)+iang];
```

```
    double nextangle=initial+actions[p];
    setangle=nextangle;
    cout<<"                    data                                                    output:
"<<RAD2DEG(velocity)<<"\t"<<RAD2DEG(angle)<<"\t"<<RAD2DEG(actions[p])<<endl;
    }
    cout<<"                    data                                                    output:
"<<RAD2DEG(velocity)<<"\t"<<RAD2DEG(angle)<<"\t"<<endl;
    j.angles[Joints::LAnklePitch]=setangle;
    j.angles[Joints::RAnklePitch]=setangle;
```