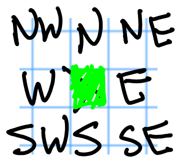
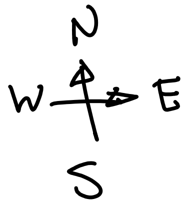
Driving Directions : "8- Connected Neighbors"





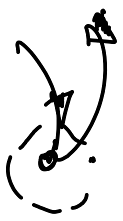
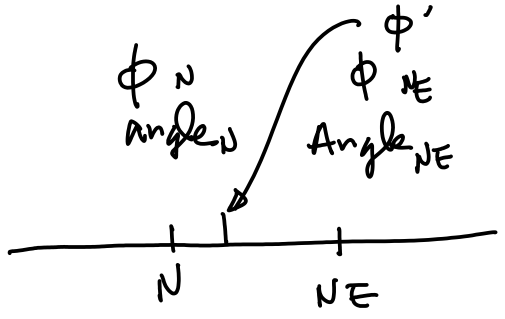


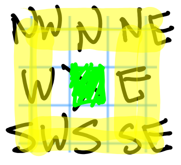
Fig.4

Find the Direction of Driving at Each step that in the end will minize the Objective Function in Eqn (9).

$$\text{Policy } \pi(\underset{\underset{\text{Action (8 Directions)}}{\uparrow}}{\alpha_{k+1}}, S_{k+1} | S_k) \rightarrow \underset{\underset{\text{Reward}}{\uparrow}}{r_{k+1}}$$

Action	Reward
N	$r_N = ?$
NW	$r_{NW} = ?$
W	$r_W = ?$
SW	$\vdots$
S	
$\vdots$	





Determine Reward Function Based ON Moving Direction of Shortest Path.

List of Possible moving Directions

1. From Fig.4. Only 5 possible Directions

NW
N
NE
W
E
SW
S
SE

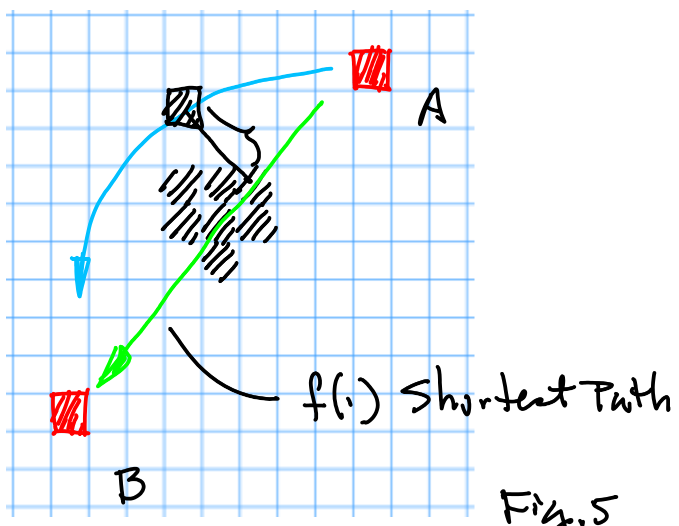


Fig.5

Example 1. 1. place 8-Direction Template (N,S,...) on top of Point A. Use shortest path, Green Line, as a reference to Define A set of Reward function Based on the direction Matching Reward (DMR) Policy

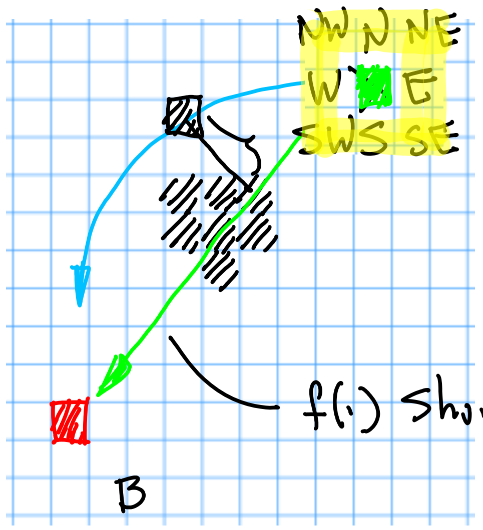


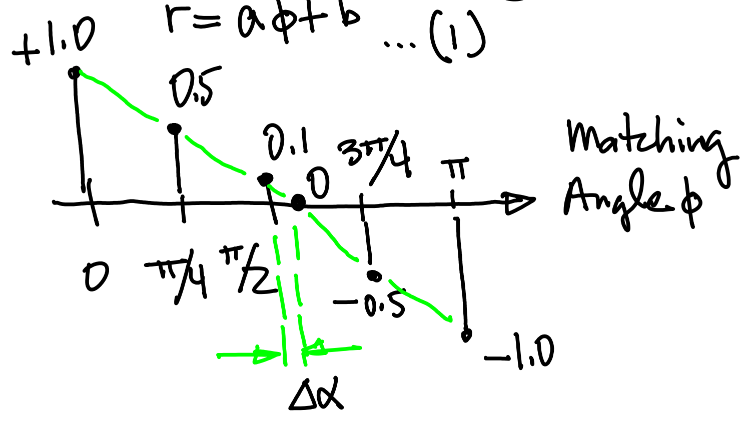
Fig.6

π_{DMR} : $\Delta = 0.5$

x-SW	+1.0	Best Matching	x-SW	overlap	
x-W	+0.6	Next Best	x-W	Angle	$< \pi/2$
x-S	+0.6	"	x-S	"	$< \pi/2$
x-SE	+0.1		x-SE	"	$< \pi/2$
x-E	-0.1	Opposite	x-E	"	$> \pi/2$
x-NW	-0.1			Angle	$> \pi/2$
x-N	-0.6			Angle	$> 3\pi/4$
x-NE	-1.0			Angle	$\simeq \pi$

Algorithm: Best Matching Direction. "Highest +" Reward
Worst matching Direction Smallest "–" Reward

$r = a\phi + b$ ... (1)



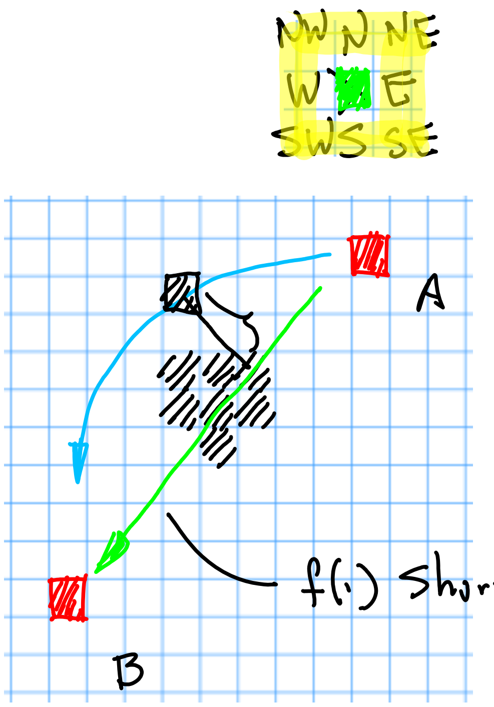


Fig.7

Program Implementation:

1° Implement Reward Function (1).

Note: Angle ϕ is formed Between Blue line and green Line.

Compute Reward.

March12(Sat),22

Step1. Setup the System As in Fig1., Fig.2 is the Abstract Form for Fig.1, So use Fig.2.

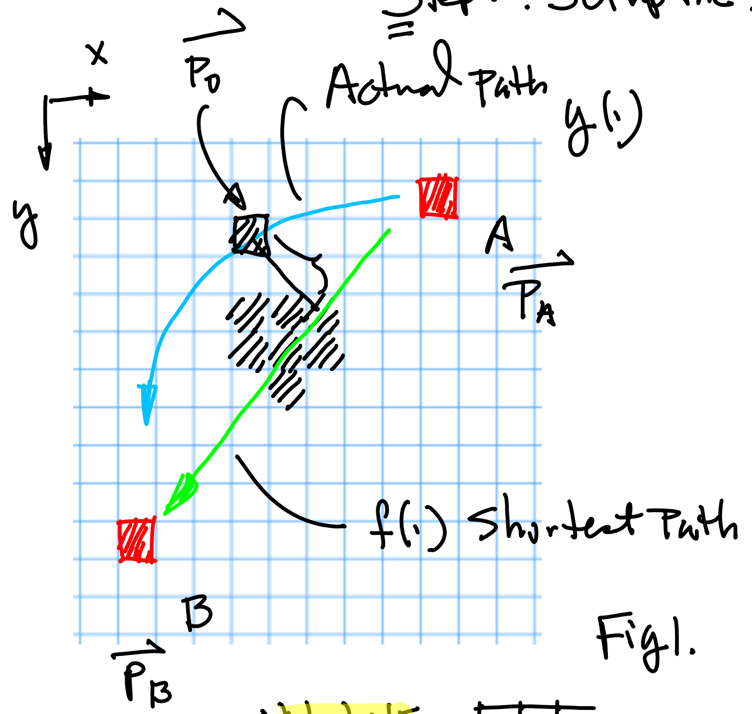


Fig1.

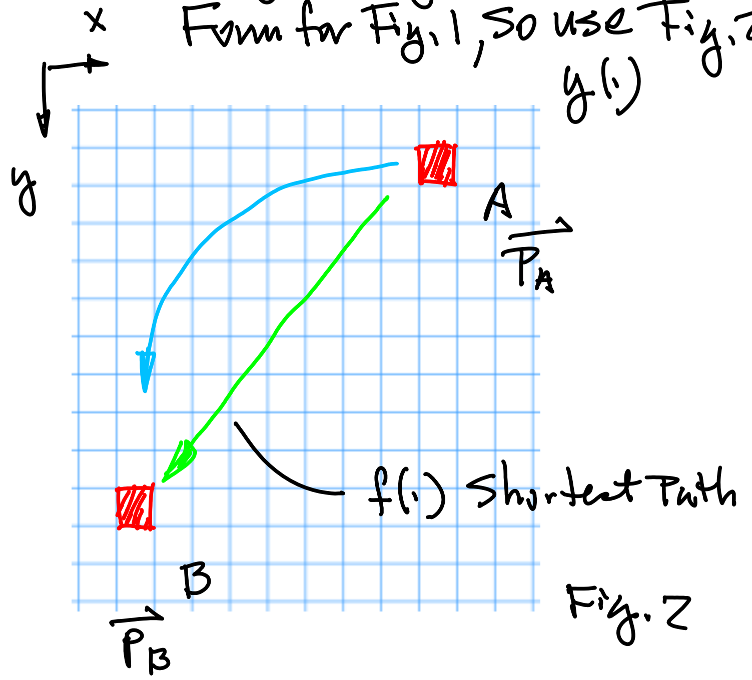


Fig.2

Step2.

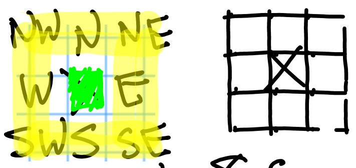


Fig2b. 8-Connected Neighbour.

Define Python module for the Angle Computation for 8-Connected Neighbours in Fig.2b.

Step3. Compute Angle Between green and Blue at location1.

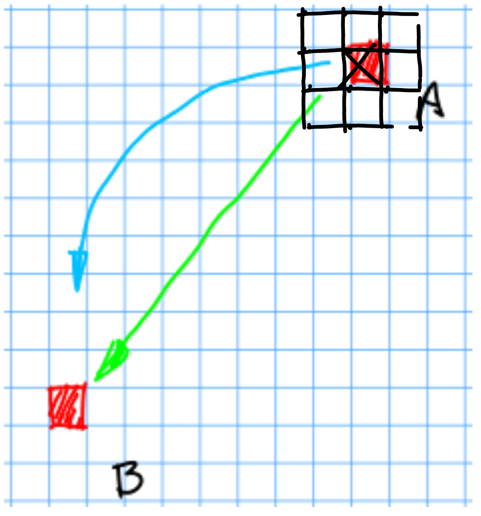


Fig.3

Compute Angle Between green and Blue at location2.

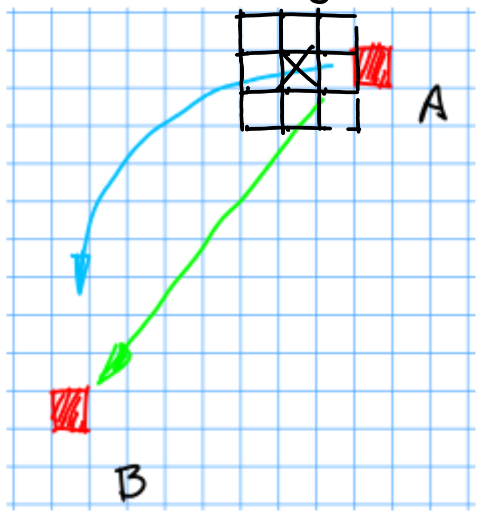


Fig.4

March 12 (Sat) 22

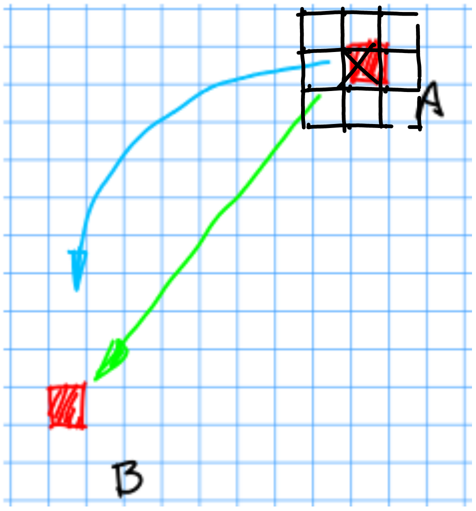


Fig.3

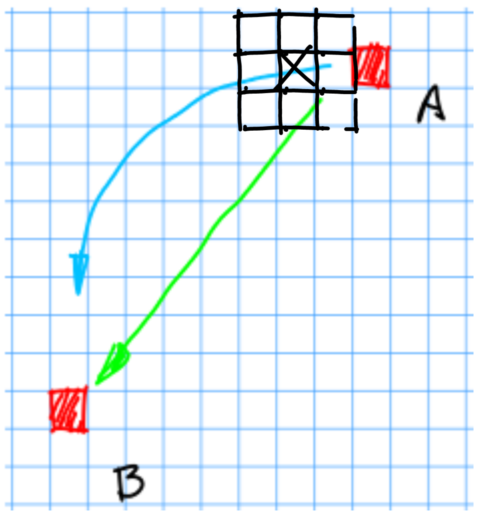


Fig.4

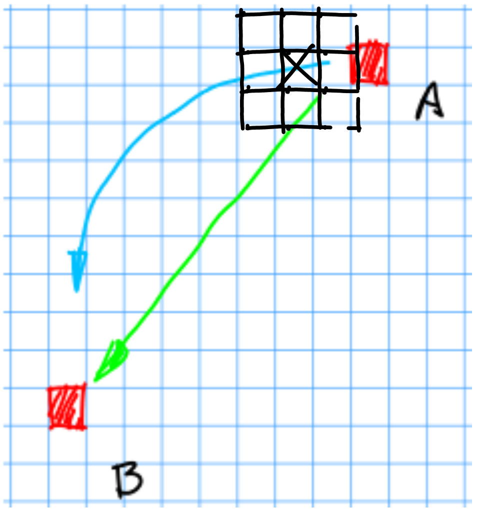


Fig5

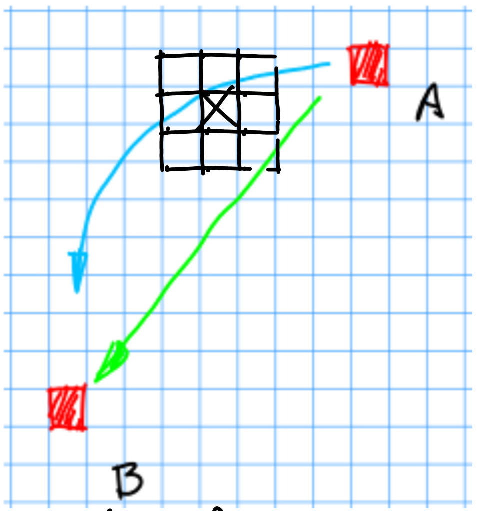


Fig.6

Compute Angle Between green and Blue at location 3.

Compute Angle Between green and Blue at location 4.

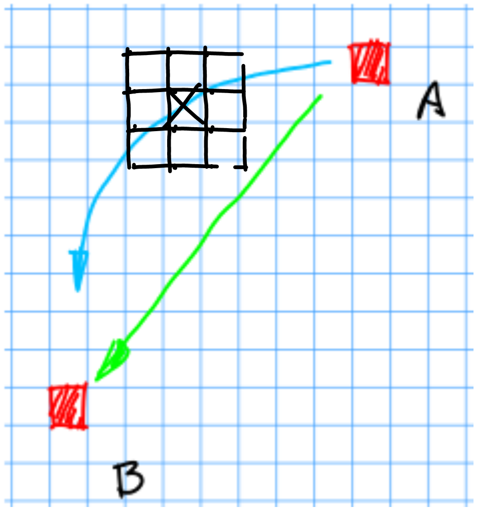


Fig 7

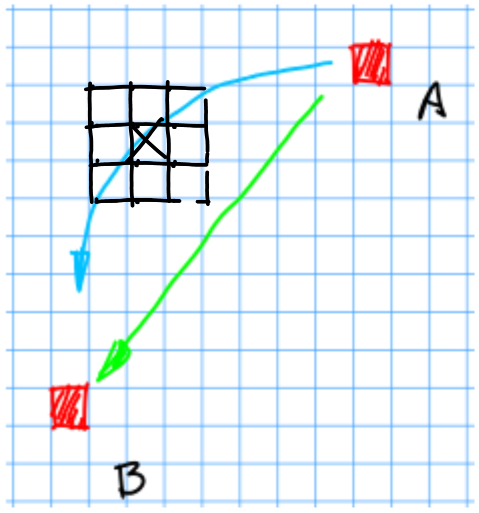


Fig. 8

Compute Angle Between green and Blue at location 5.

Compute Angle Between green and Blue at location 6.

March 12 (Sat), 22

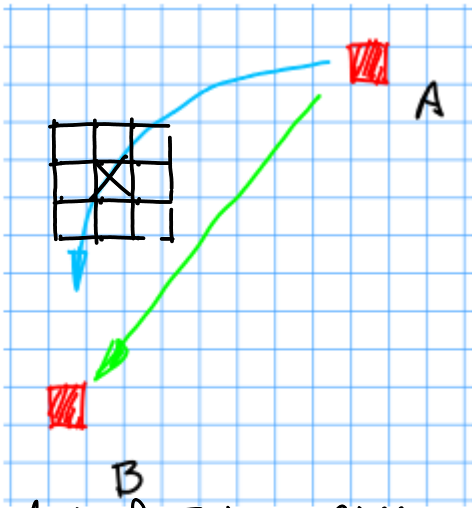


Fig 9

Compute Angle Between green and Blue at location 7.

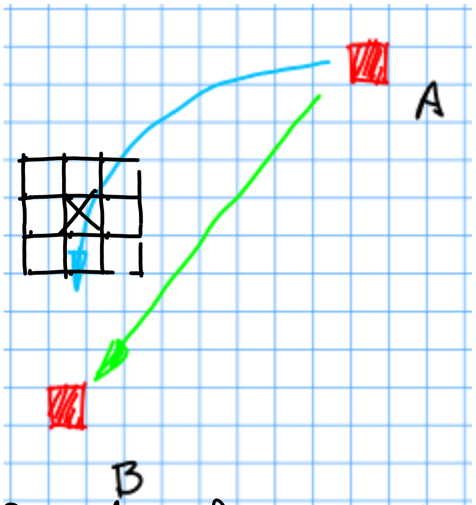


Fig. 10

Compute Angle Between green and Blue at location 8.

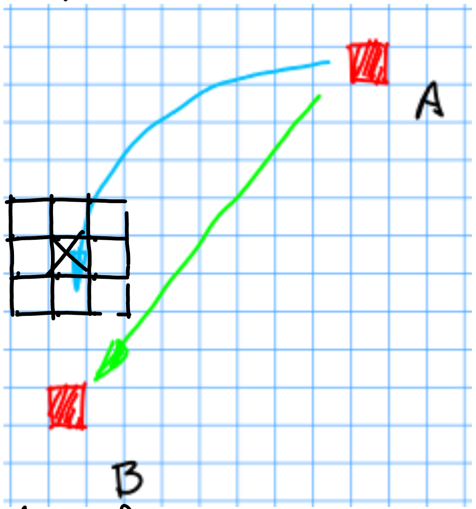


Fig 11

Compute Angle Between green and Blue at location 9.

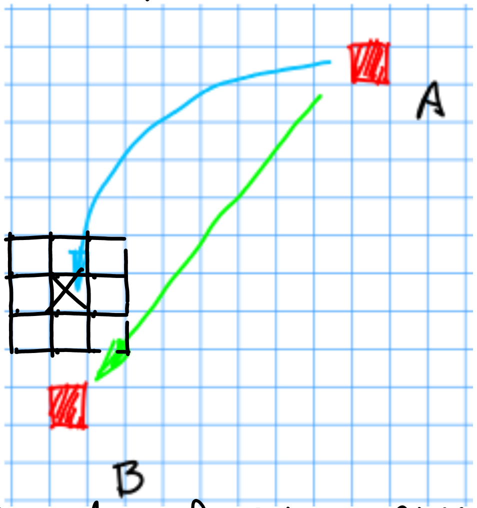


Fig. 12

Compute Angle Between green and Blue at location 10.

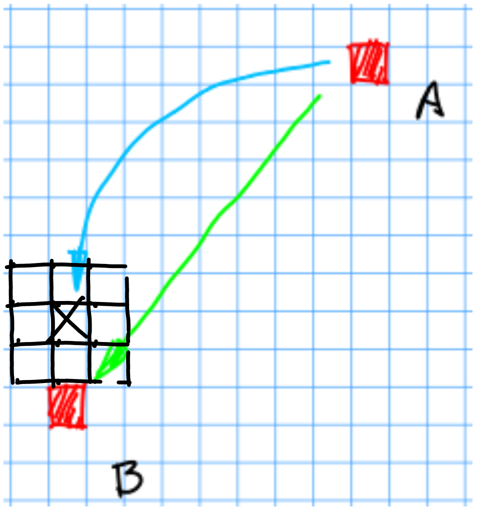


Fig 13

Compute Angle at location 11.

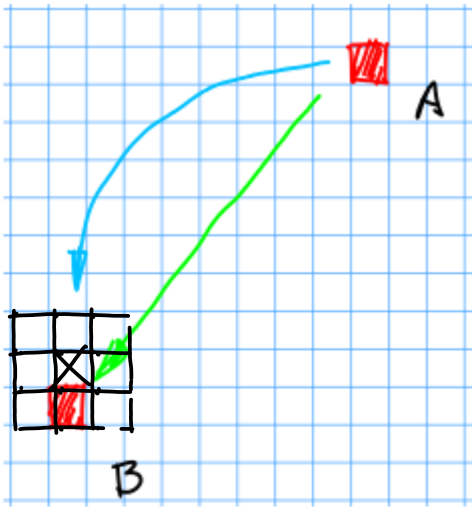


Fig. 14

Compute Angle at location 12.

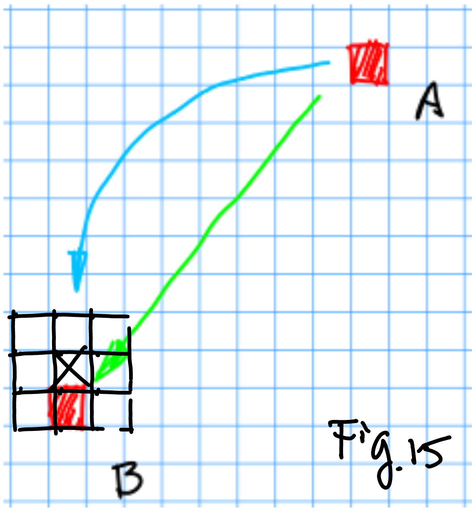


Fig. 15

Compute Angle at location 13.

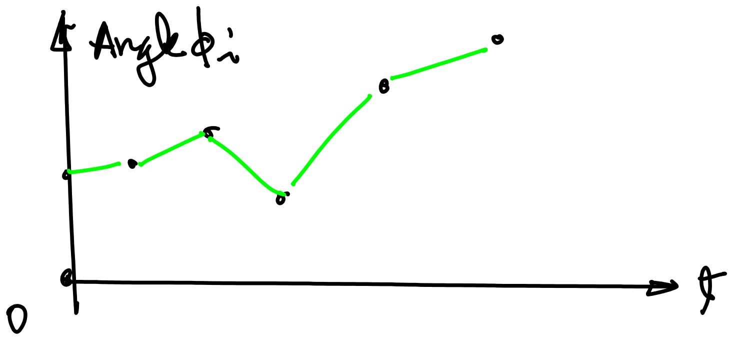
March 12 (Sat), 22

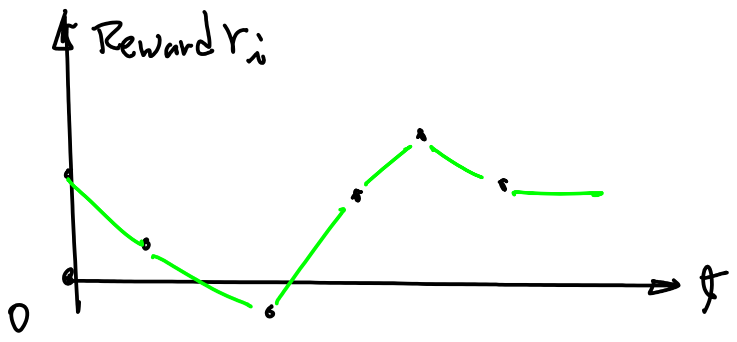
Step 4

Plot All the Angles $\phi_1, \phi_2, \cdots, \phi_i \cdots$ in the plot below, plot All Reward Function Values $r_1, r_2, \cdots, r_i \cdots$ in the plot below.

Then find Sum of all Rewards.

$$R_\Sigma = \sum_{i=1}^{N-1} r_i \quad \cdots (1)$$





March 13th (Sun) with B.P.

(End)

$\vec{x} \times \vec{y} = \vec{z}$

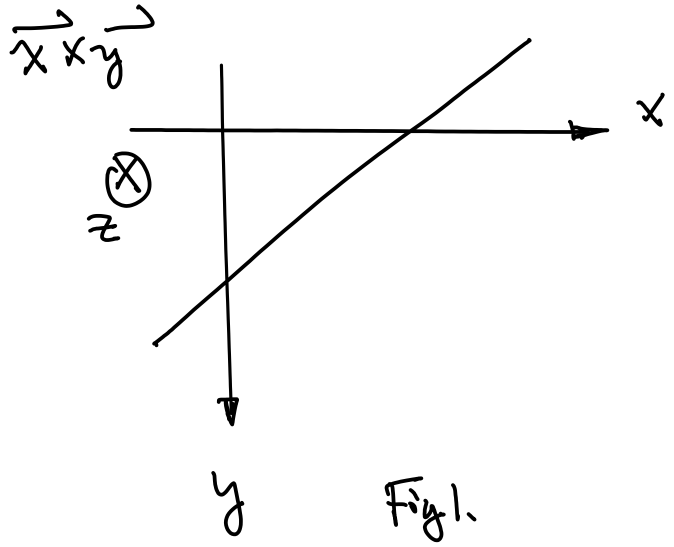
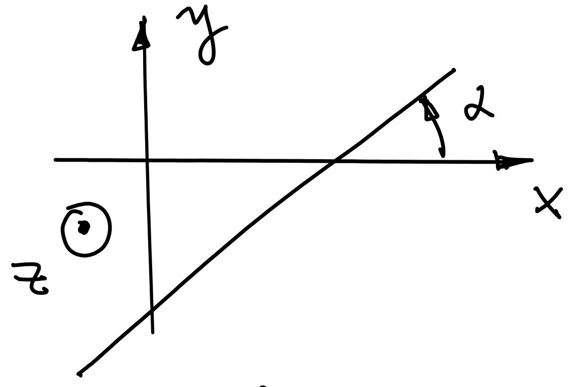


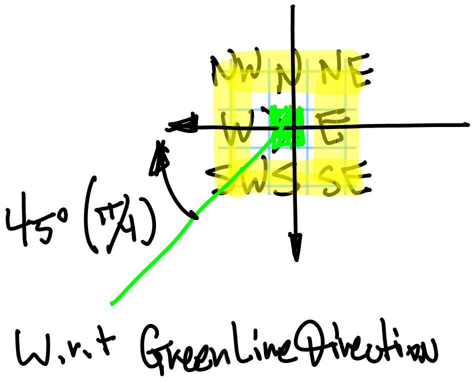
Fig 1.

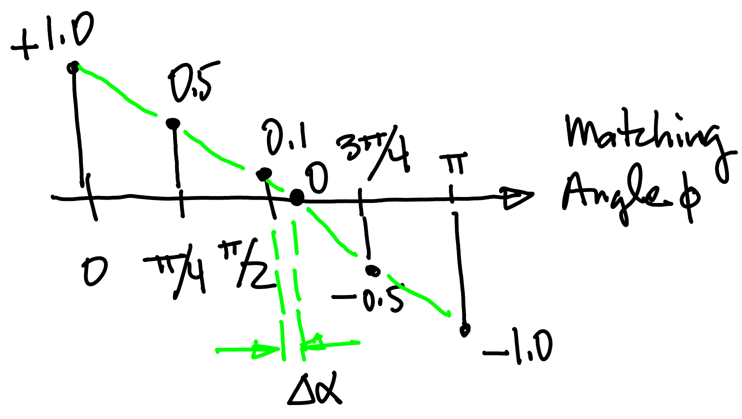


Guideline

Fig. 2

No "World coordinate System" yet





π_{DMR} : $\Delta = 0.5$

x-SW	+1.0	Best matching	x-SW overlap
x-W	+0.6	Next Best	x-W Angle < π/2
x-S	+0.6	" "	x-S " < π/2
x-SE	+0.1		x-SE " < π/2
x-E	-0.1	Opposite	x-E " > π/2
x-NW	-0.1		Angle > π/2
x-N	-0.6		Angle > 3π/4
x-NE	-1.0		Angle ≃ π

March 14 (monday), with. YY, BP.

robotics-open_abb / aiv200 / 190g-deep-reinforcement-learning / **190g-3-6DoF-Action-State-Reward-SS-2021-03-17.pdf**

6 DoF Robot Unity

How to train your Robot Arm?. Training a 6 axis robot arm using Unity… | by Raju K | XRPractices | Medium

rkandas/RobotArmMLAgentUnity: Training 6 axis robot arm Inverse kinematics using Unity ML Agents (github.com)

1. Actions: An array of actions – each action in the array represents the degree of rotation. We have 5 types of actions in total: 1 Rotate and 4 Bends.

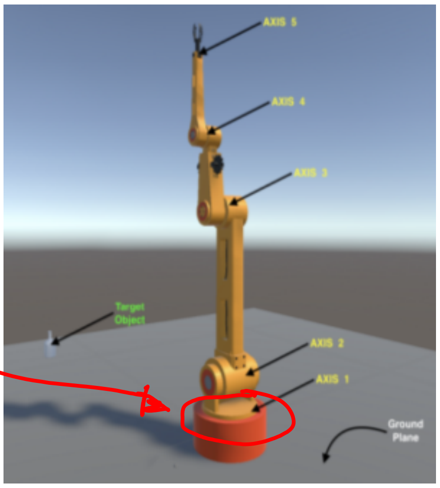
1.1. Axis 1: is the bottom-most axis and can rotate 0 to 360 degrees [Rotate]

```
armAxes[0].transform.localRotation =
Quaternion.AngleAxis(angles[0] * 180f, armAxes[0].GetComponent<Axis>().rotationAxis);
```

CTI ONE march 14, 22

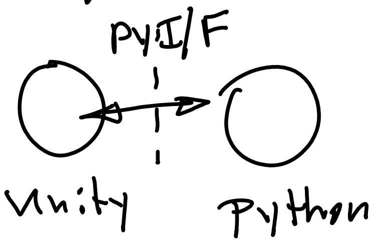
1.1. Axis 1: is the bottom-most axis and can rotate 0 to 360 degrees [Rotate]

```
armAxes[0].transform.localRotation =
Quaternion.AngleAxis(angles[0] * 180f, armAxes[0].GetComponent<Axis>().rotationAxis);
```



C# { a. Physical model (Dimension)
b. Physics of the model. { Rotation direction
└ Angle
c. Graphical model

{ Step 1. a–c : Move OpenCV model to Unity from CTI ONE model, And the implementation code is from CTI One Team, especially from Mr. Yusuke Yakura.
└ Step 2. C# ML Interface



OpenCV { 1. Angle
2. Index
└3. Reward (I)

March 16 (Wed)

1. Verification of YY's Implementation

2. Provide Hand Calculation.

① from start position to the end position.

	Position	Angle	Distance	Reward
$\vec{P_A}$	(9,1)	Φ_i	d_i L_i	$r(\Phi_i, L_i)$
	⋮		⋮	
$\vec{P_i}$	(4,2)			
	⋮			
$\vec{P_B}$	(1,10)			

y

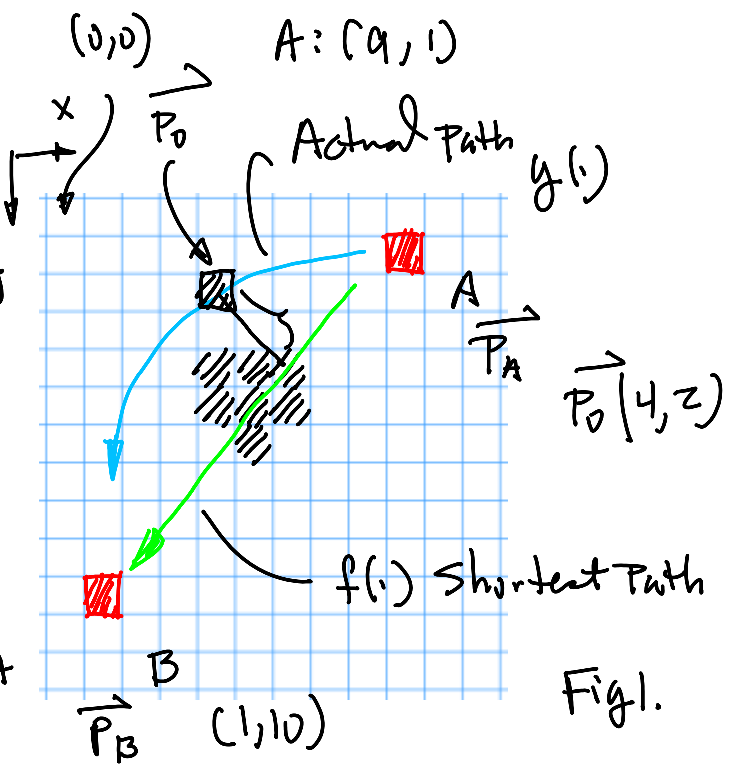


Fig 1.

② Record Heuristic Motion, Path.txt

③ Run GTI One Version 0.1 Code,
Make Comparison for Verification.