EE4308

Part 2 Project Briefing and Tutorial

Dr. Phang Swee King (skphang@nus.edu.sg)

Temasek Laboratories

National University of Singapore

Class schedule

Week 9

15-03-2016 (Wednesday) – 1000 to 1200 – E4-04-07

- Part 2 Project Briefing & Tutorial

17-03-2016 (Friday) – 1400 to 1500 – E4-04-07

- Free!

Week 10-12

Every Wednesday – 1000 to 1200 – T-Lab #04-02

- Optional consultation for 2 teams (email appointment)

Every Friday – 1400 to 1500 – T-Lab #04-02

- Optional consultation for 1 team (email appointment)

Email: skphang@nus.edu.sg

Are you prepared?

Have you read the project manual?

- ✓ Understand the experiments requirements
- ✓ Prepare sufficient materials for the project

What skills do you need?

- ✓ Understand basic autonomous control and path planning of UAVs
- ✓ Basic C++/MATLAB programming language
- ✓ Understand ROS and able to create small programs
- ✓ Teamwork!

Are you prepared?

Found your teammates?

- ✓ Group of 6 (Maximum 5 groups)
- ✓ Prefer to have similar time-table for this semester
- ✓ Work together (own time) on ROS simulation
- ✓ Work together for 2 × 2-hour sessions on practical experiment

Register your group at IVLE!!

To use ROS desktop (Task 1 & 3)

Week 9/10		
Monday	0900-1200	1400-1700
Tuesday	0900-1200	1400-1700
Wednesday	0900-1200	1400-1700
Thursday	0900-1200	1400-1700
Friday	0900-1200	1400-1700

Week 11		
Monday (27/3)		1400-1700
Tuesday (28/3)		1400-1700
Wednesday (29/3)	0900-1200	1400-1700
Thursday (30/3)	0900-1200	1400-1700
Friday (31/3)		

Week 12		
Monday (3/4)		1400-1700
Tuesday (4/4)		1400-1700
Wednesday (5/4)	0900-1200	1400-1700
Thursday (6/4)	0900-1200	1400-1700
Friday (7/4)		

Location: M&A Lab, E4A-03-04

How to book for slot?

- ✓ No booking is needed, just go during free time (as stated in the schedule above)

To use VICON room (Task 2 & 3)

Week 10	Slot 1	Slot 2
Monday (20/3)	1400-1600	1600-1800
Tuesday (21/3)	1400-1600	1600-1800
Wednesday (22/3)	1400-1600	1600-1800
Thursday (23/3)	1400-1600	1600-1800
Friday (24/3)	1330-1530	1530-1730

Week 11	Slot 1	Slot 2
Monday (27/3)	1400-1600	1600-1800
Tuesday (28/3)	1400-1600	1600-1800
Wednesday (29/3)	1400-1600	1600-1800
Thursday (30/3)	1400-1600	1600-1800
Friday (31/3)	1330-1530	1530-1730

Week 12	Slot 1	Slot 2
Monday (3/4)	1400-1600	1600-1800
Tuesday (4/4)	1400-1600	1600-1800
Wednesday (5/4)	1400-1600	1600-1800
Thursday (6/4)	1400-1600	1600-1800
Friday (7/4)	1330-1530	1530-1730

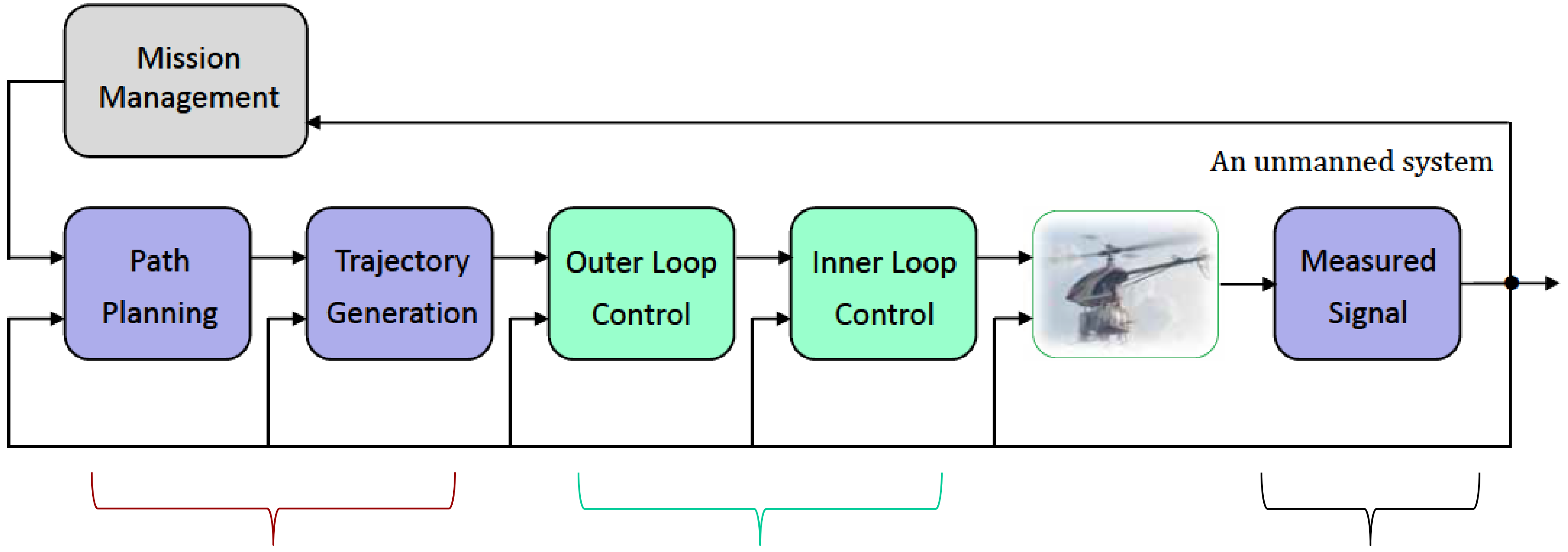
Location: T-Lab Level 4 (#04-02)

How to book for VICON room slot?

- ✓ Email skphang@nus.edu.sg for 2 preferred slots
- ✓ First come first serve

System overview

Control loops



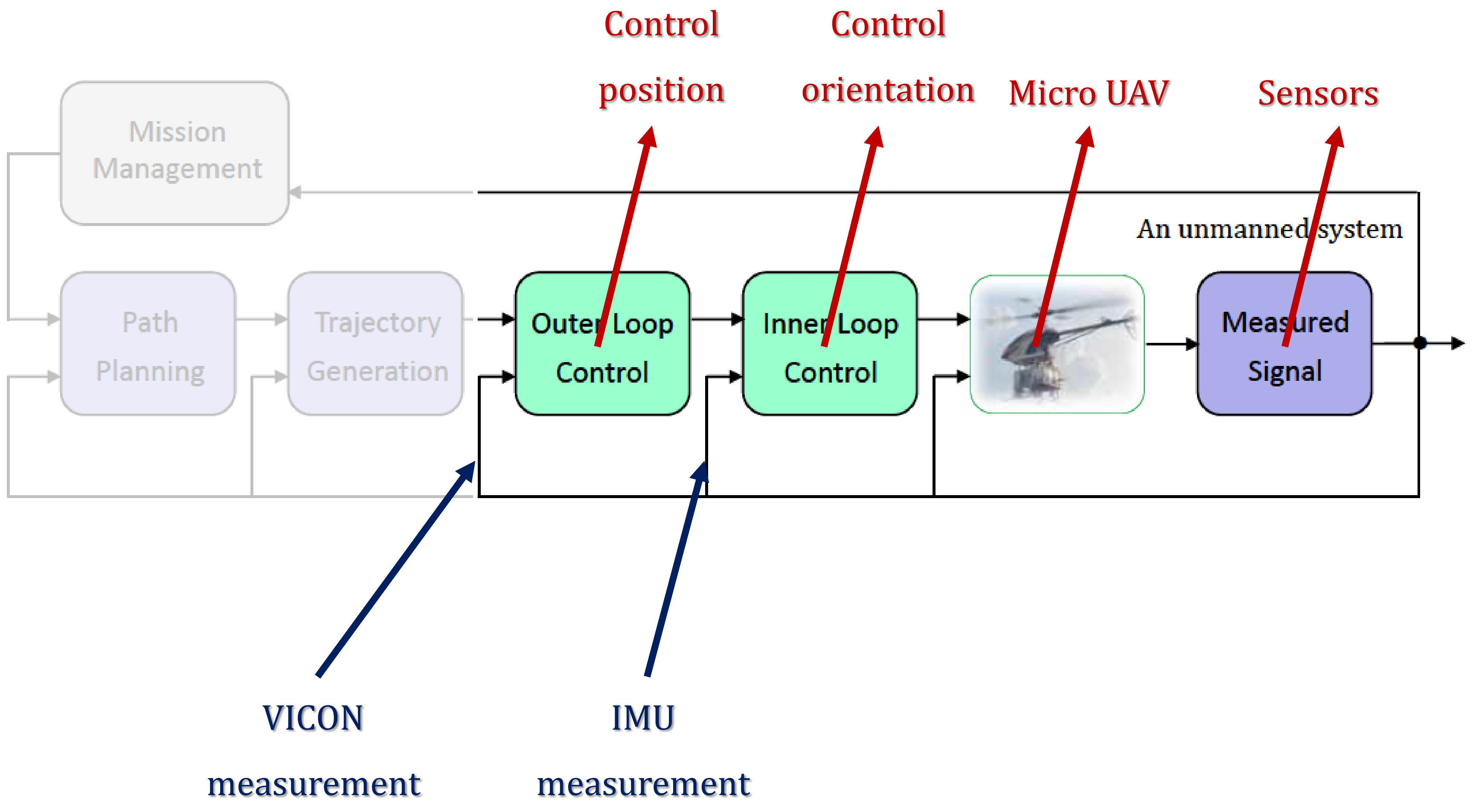
- **You will design this!!**

- Tuned and implemented in on-board processor

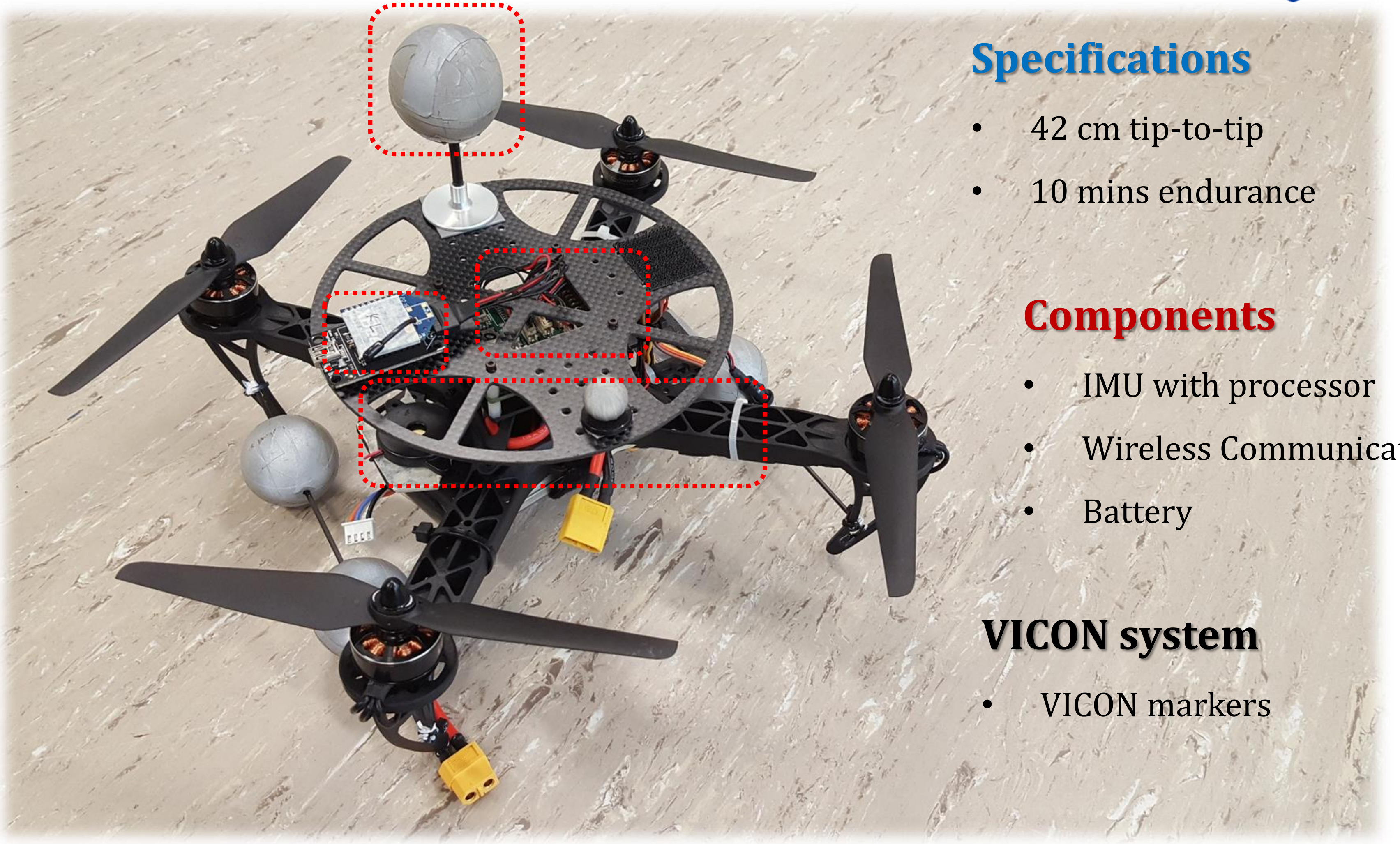
- Sensors from IMU
- VICON system

System overview

Control loops



Micro UAV



Specifications

- 42 cm tip-to-tip
- 10 mins endurance

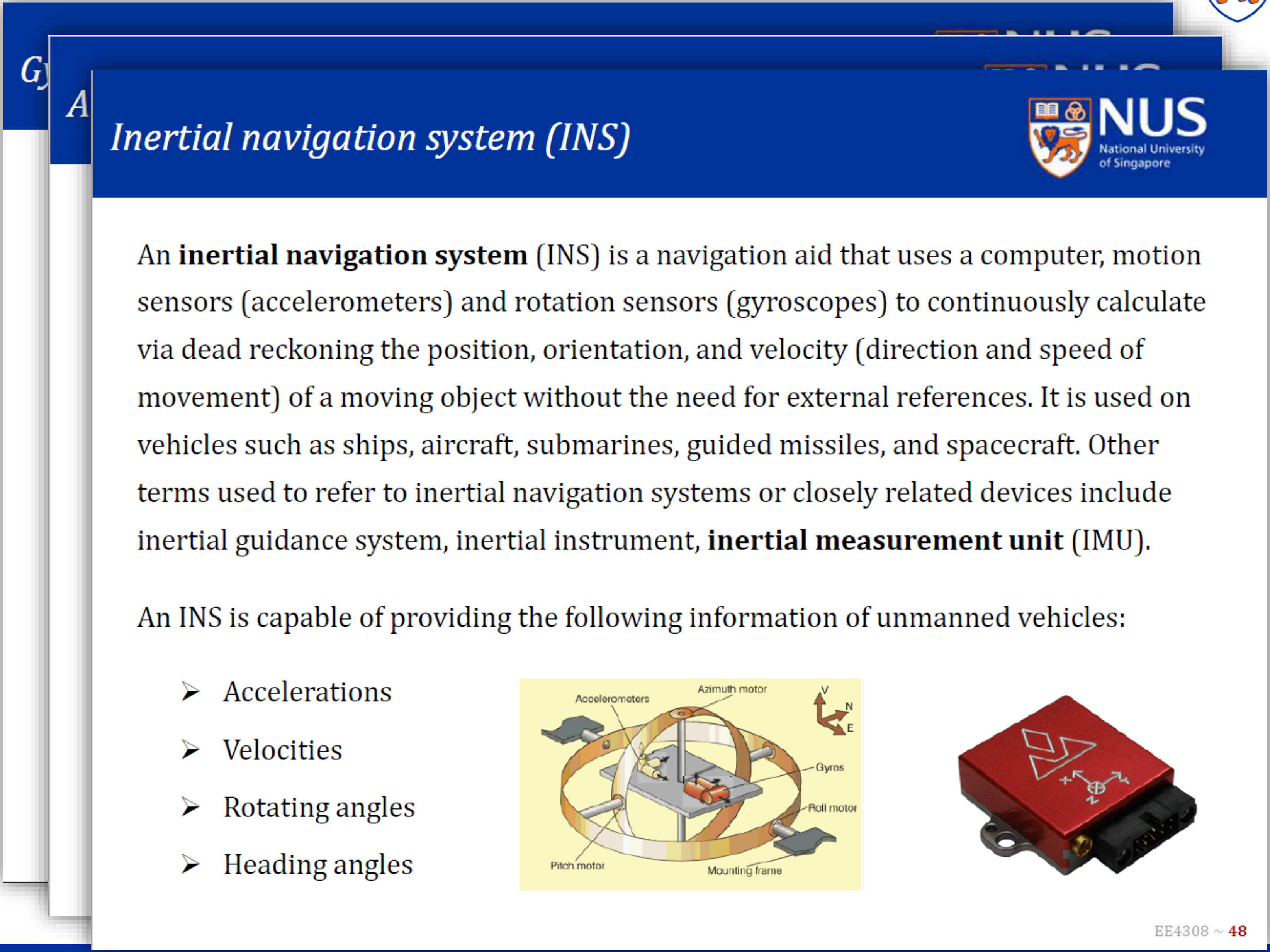
Components

- IMU with processor
- Wireless Communication
- Battery

VICON system

- VICON markers

Sensors – inner-loop

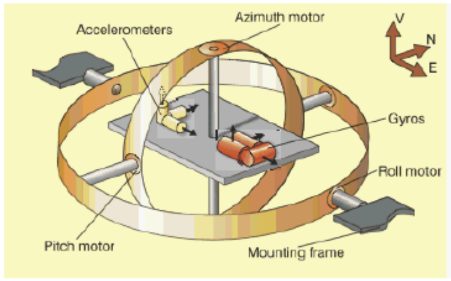


Inertial navigation system (INS)

An **inertial navigation system** (INS) is a navigation aid that uses a computer, motion sensors (accelerometers) and rotation sensors (gyroscopes) to continuously calculate via dead reckoning the position, orientation, and velocity (direction and speed of movement) of a moving object without the need for external references. It is used on vehicles such as ships, aircraft, submarines, guided missiles, and spacecraft. Other terms used to refer to inertial navigation systems or closely related devices include inertial guidance system, inertial instrument, **inertial measurement unit** (IMU).

An INS is capable of providing the following information of unmanned vehicles:

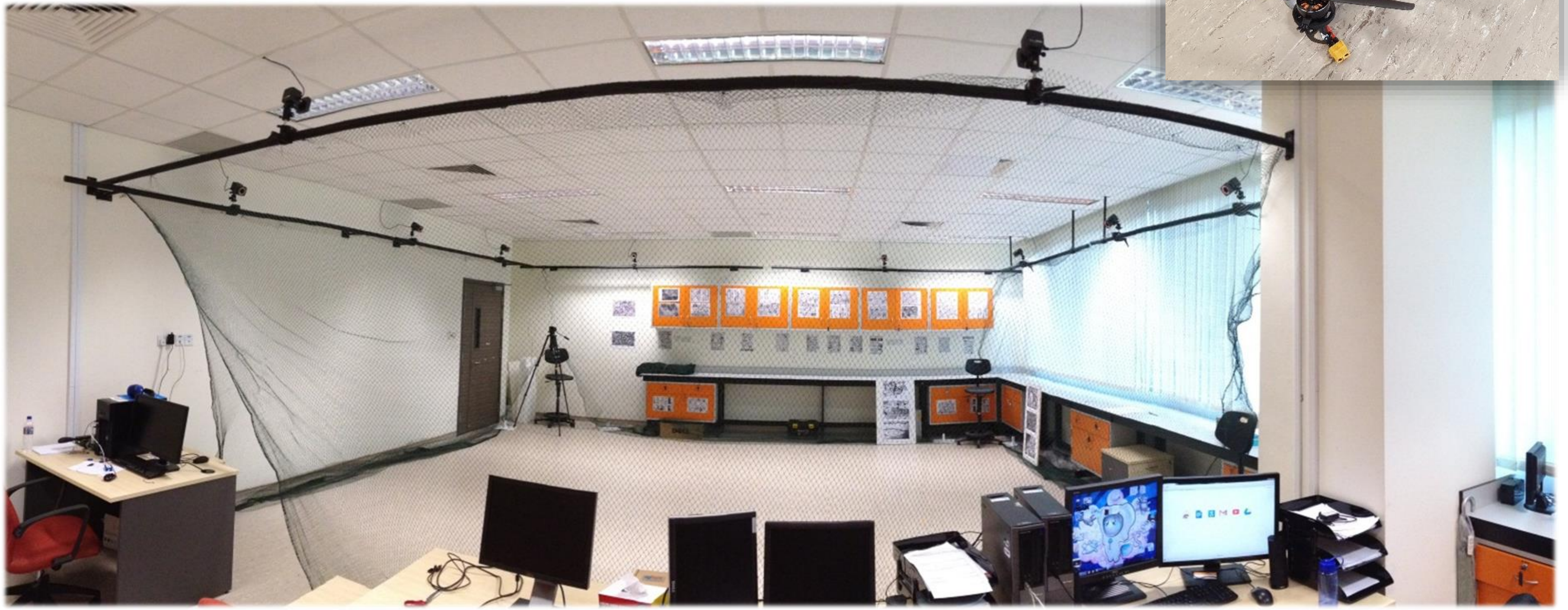
- Accelerations
- Velocities
- Rotating angles
- Heading angles



Sensors – outer-loop

Motion capture system is utilized for the indoor tests of UAVs

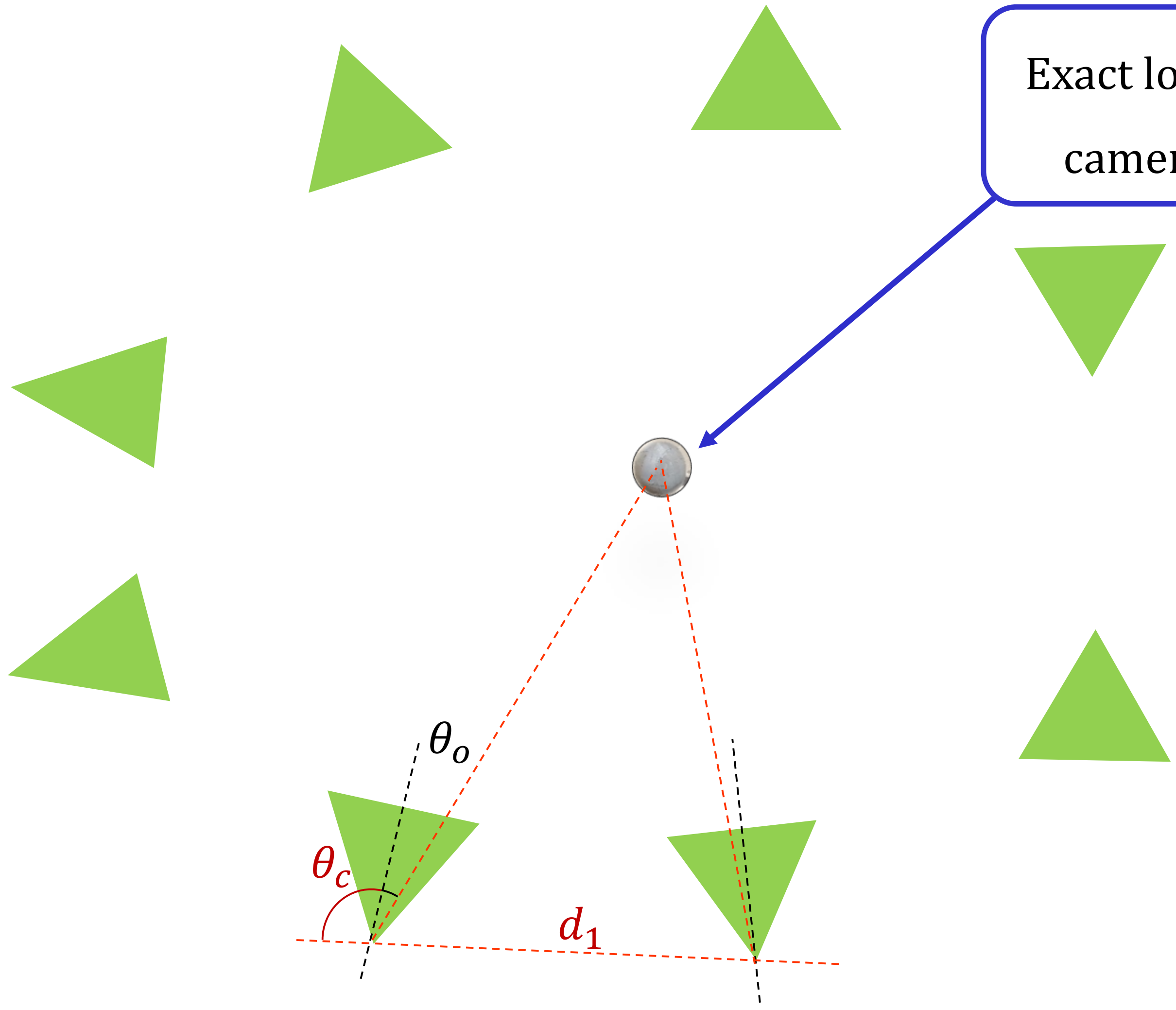
- ✓ Dynamic modeling
- ✓ Control verification
- ✓ Sensor calibration



Sensors – outer-loop

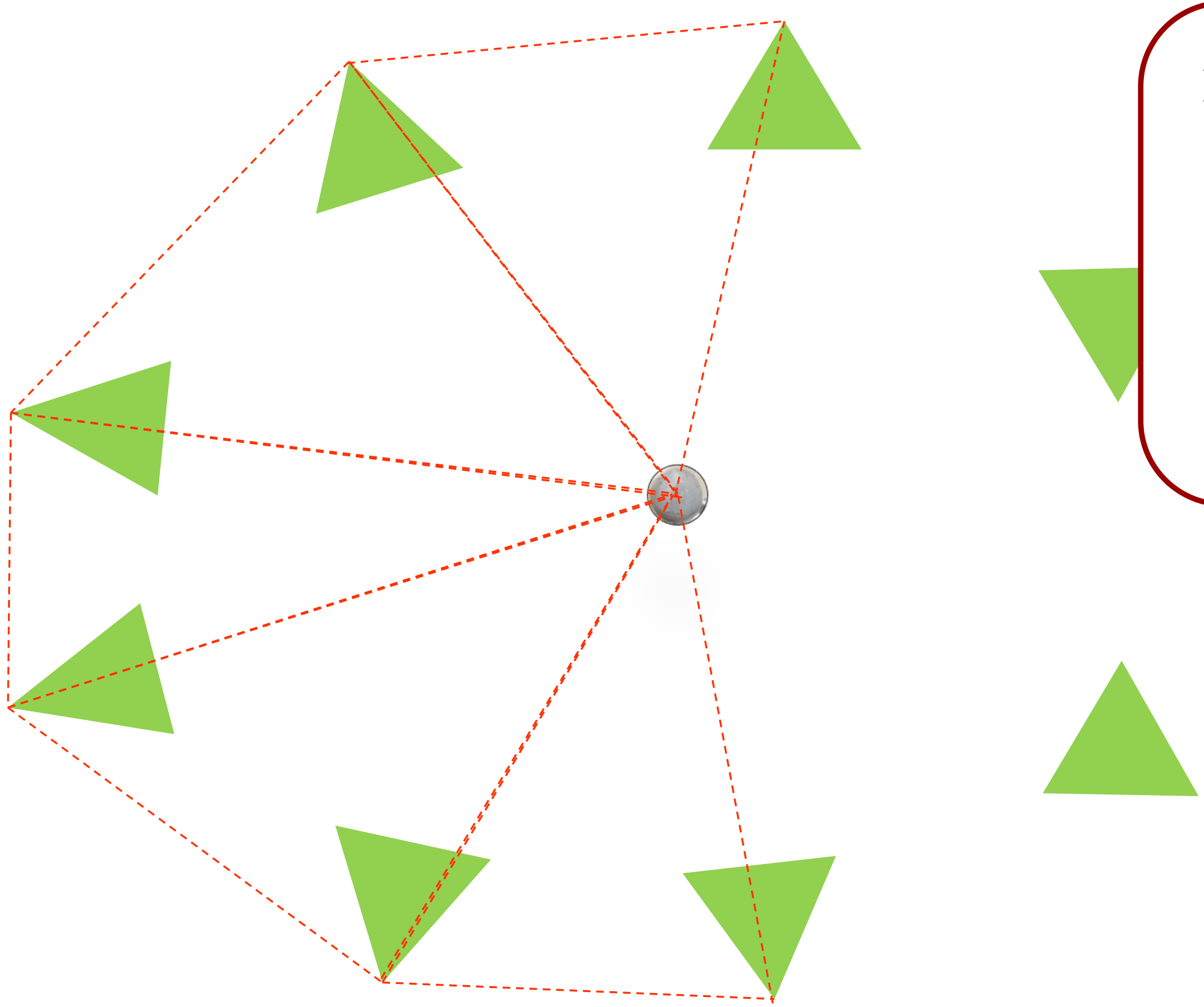
How to estimate position with external camera?

Exact location can be calculated if cameras position are known!!



Sensors – outer-loop

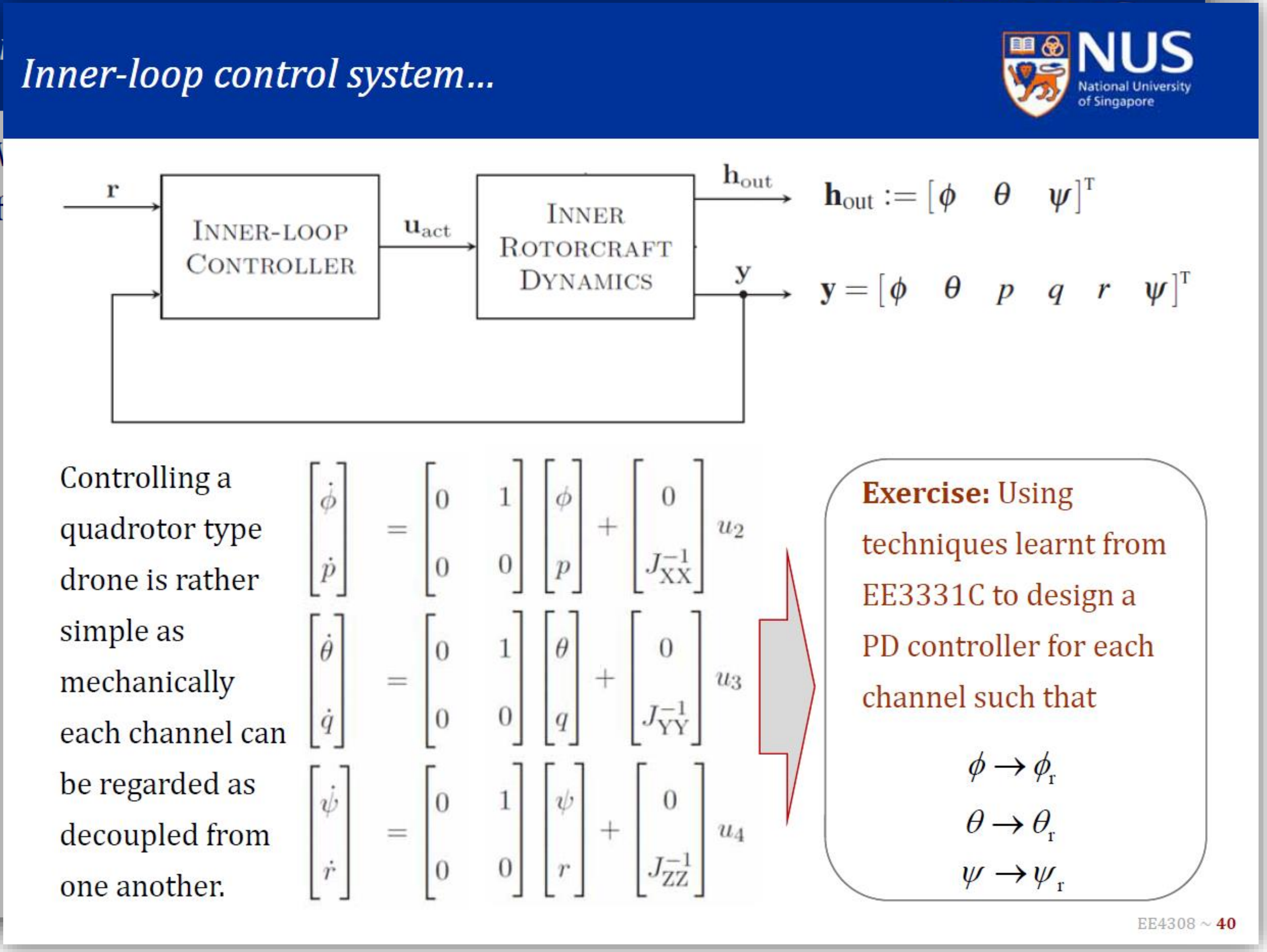
How to estimate position with external camera?



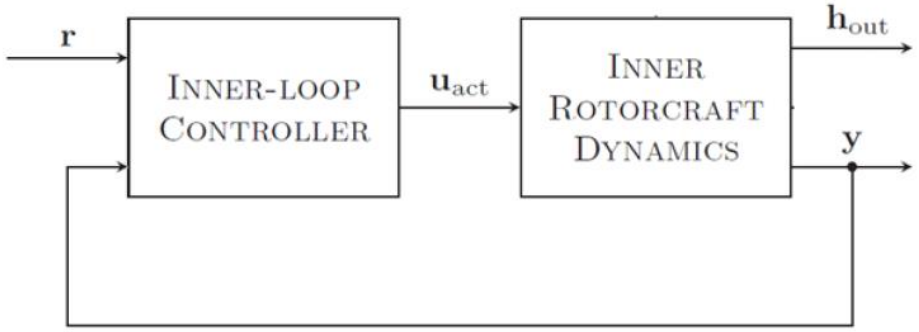
More accurate position can be estimated by averaging the position calculated by each pair of camera

3 cameras used for each estimation

Control – inner-loop



Inner-loop control system…


r
INNER-LOOP CONTROLLER
$\mathbf{u}_{\text{act}}$
INNER ROTORCRAFT DYNAMICS
$\mathbf{h}_{\text{out}}$
$\mathbf{y}$

$$\mathbf{h}_{\text{out}} := [\phi \quad \theta \quad \psi]^{\mathrm{T}}$$

$$\mathbf{y} = [\phi \quad \theta \quad p \quad q \quad r \quad \psi]^{\mathrm{T}}$$

Controlling a quadrotor type drone is rather simple as mechanically each channel can be regarded as decoupled from one another.

$$\begin{bmatrix}\dot\phi\\ \dot p\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\phi\\ p\end{bmatrix} + \begin{bmatrix}0\\ J_{\mathrm{XX}}^{-1}\end{bmatrix} u_2$$

$$\begin{bmatrix}\dot\theta\\ \dot q\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\theta\\ q\end{bmatrix} + \begin{bmatrix}0\\ J_{\mathrm{YY}}^{-1}\end{bmatrix} u_3$$

$$\begin{bmatrix}\dot\psi\\ \dot r\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\psi\\ r\end{bmatrix} + \begin{bmatrix}0\\ J_{\mathrm{ZZ}}^{-1}\end{bmatrix} u_4$$

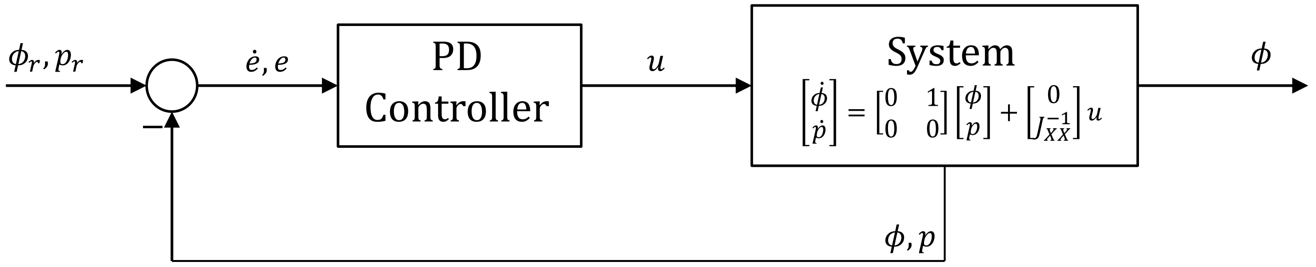
Exercise: Using techniques learnt from EE3331C to design a PD controller for each channel such that

$$\phi \to \phi_{\mathrm{r}}$$
$$\theta \to \theta_{\mathrm{r}}$$
$$\psi \to \psi_{\mathrm{r}}$$

Control – inner-loop

The system
$$\begin{bmatrix}\dot\phi\\ \dot p\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\phi\\ p\end{bmatrix} + \begin{bmatrix}0\\ J_{XX}^{-1}\end{bmatrix}u$$

The output
$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}\phi\\ p\end{bmatrix}$$


ϕ_r, p_r
$-$
$\dot e, e$
PD Controller
u
System
$\begin{bmatrix}\dot\phi\\ \dot p\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\phi\\ p\end{bmatrix} + \begin{bmatrix}0\\ J_{XX}^{-1}\end{bmatrix}u$
ϕ
ϕ, p

Errors
$$\begin{cases}\phi_e = \phi_r - \phi\\ p_e = p_r - p\end{cases} \rightarrow \begin{matrix} e = \phi_e\\ \dot e = \dot\phi_e = p_e\end{matrix}$$

PD controller
$$u = k_D\dot e + k_P e$$

Control – inner-loop

The system

$$\underbrace{\begin{bmatrix}\dot{\phi}\\ \dot{p}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\phi\\ p\end{bmatrix} + \begin{bmatrix}0\\ J_{XX}^{-1}\end{bmatrix}u}_{\underbrace{\begin{array}{c}\dot{\phi} = p\\ \dot{p} = J_{XX}^{-1}u\end{array}}_{\ddot{\phi} = J_{XX}^{-1}u}} \qquad \underbrace{y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}\phi\\ p\end{bmatrix}}_{\underbrace{y = \phi}_{}}$$

$$\ddot{\phi} = J_{XX}^{-1}u \longrightarrow \ddot{y} = J_{XX}^{-1}u$$

Transfer function

$$s^2Y(s) = J_{XX}^{-1}U(s) \quad \rightarrow \quad G(s) = \frac{Y(s)}{U(s)} = \frac{1}{J_{XX}s^2}$$

PD controller

$$u = k_D\dot{e} + k_Pe$$

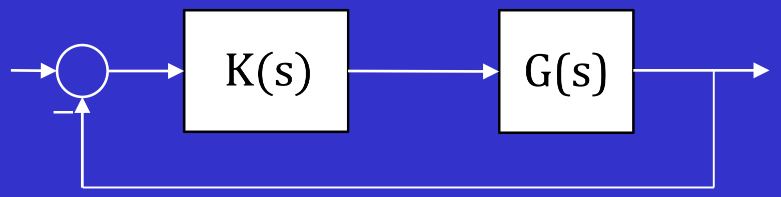
$$U(s) = k_DsE(s) + k_PE(s) \quad \rightarrow \quad K(s) = \frac{U(s)}{E(s)} = k_P + k_Ds$$

Control – inner-loop

Closed-loop transfer function

$$H(s) = \frac{\frac{k_P + k_D s}{J_{XX} s^2}}{1 + \frac{k_P + k_D s}{J_{XX} s^2}}$$

$$H(s) = \frac{k_D s + k_P}{J_{XX} s^2 + k_D s + k_P}$$



$$H(s) = \frac{KG}{1 + KG}$$

Final value theorem (FVT)

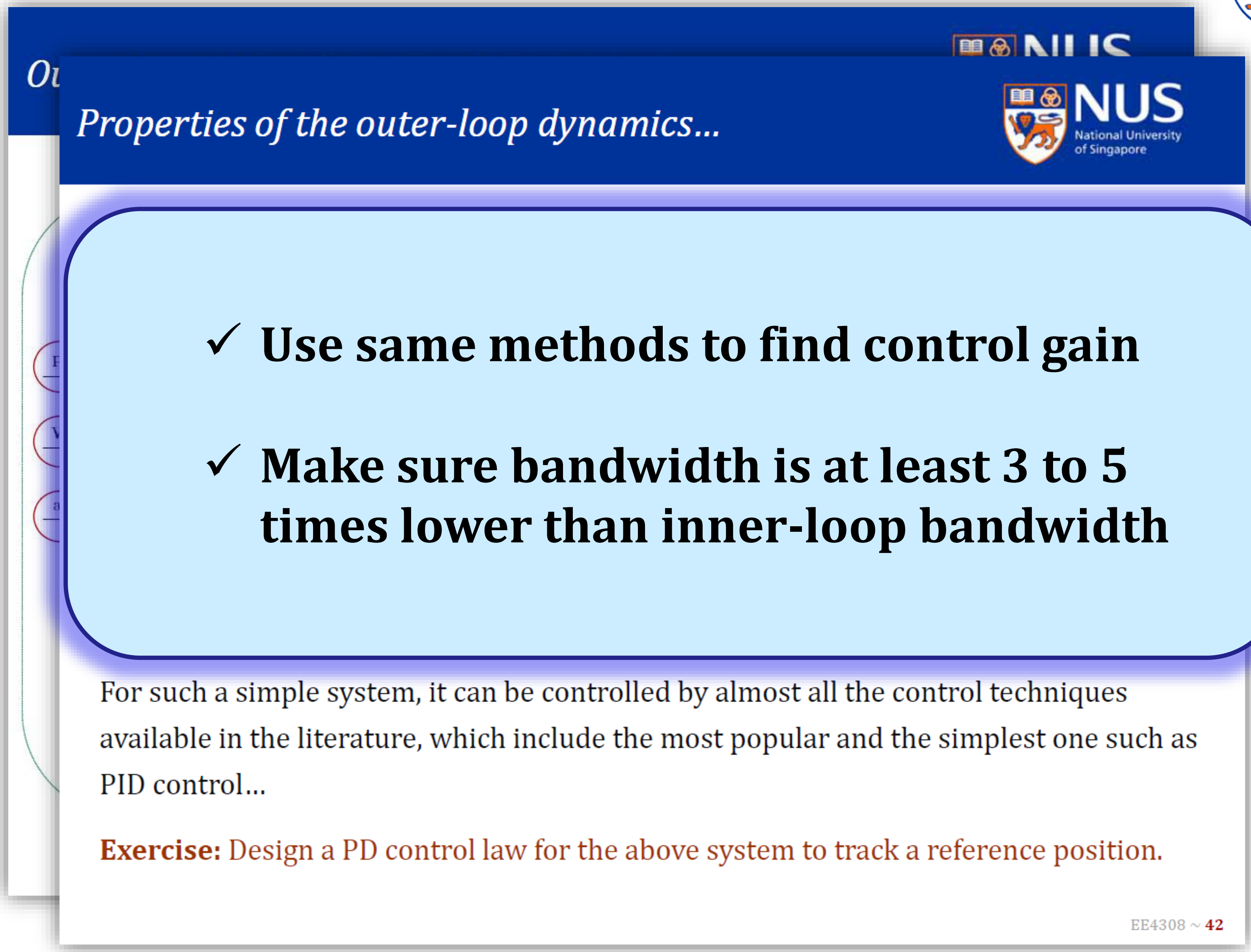
$$\frac{Y(s)}{R(s)} = \frac{k_D s + k_P}{J_{XX} s^2 + k_D s + k_P}$$

Step-response

$$Y(s) = \frac{k_D s + k_P}{J_{XX} s^2 + k_D s + k_P}\left(\frac{r}{s}\right)$$

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} \frac{k_D s + k_P}{J_{XX} s^2 + k_D s + k_P}\left(\frac{r}{s}\right) s = r$$

Control – outer-loop



Properties of the outer-loop dynamics…

For such a simple system, it can be controlled by almost all the control techniques available in the literature, which include the most popular and the simplest one such as PID control…

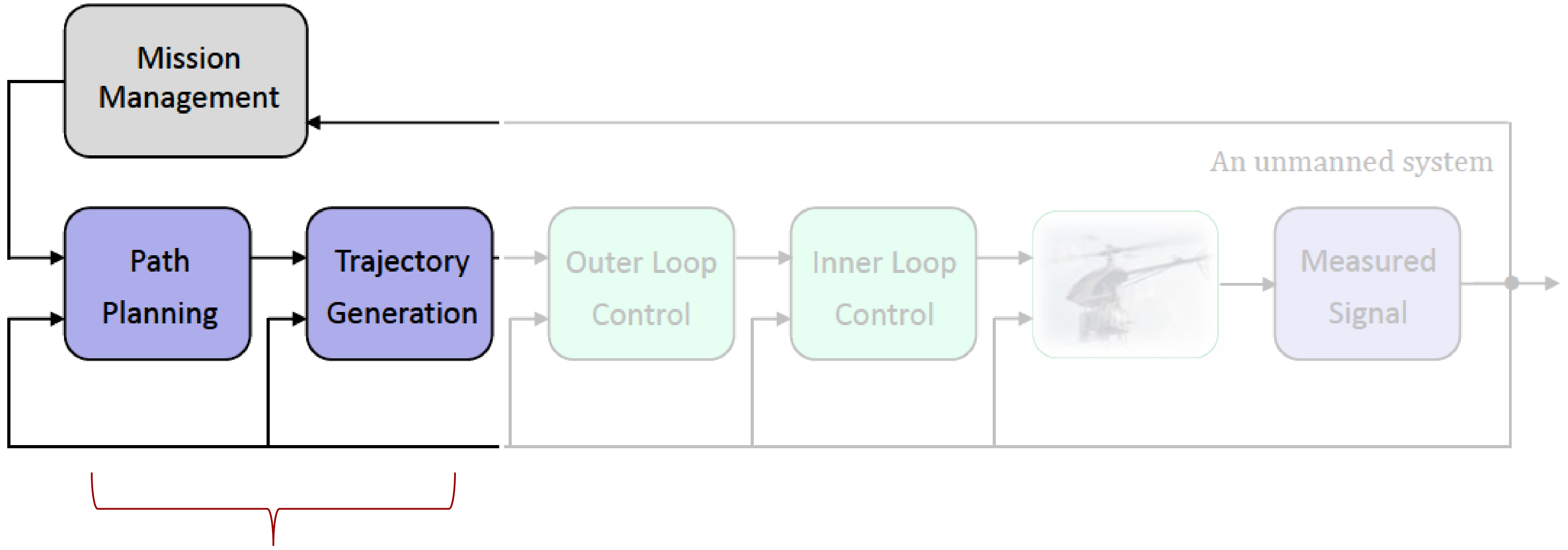
Exercise: Design a PD control law for the above system to track a reference position.

EE4308 ~ **42**

- ✓ **Use same methods to find control gain**
- ✓ **Make sure bandwidth is at least 3 to 5 times lower than inner-loop bandwidth**

System overview

Control loops



- **You will design this!!**

Project overview

Task 1

- Path planning or design of a trajectory for UAV to travel in a fixed confined space
- Make a simulator to show clearly the trajectory of UAV
- UAV must avoid all obstacles and the behavior should be shown in the simulator

15%

Task 2

- With the path designed in Task 1, implement and run it in an actual UAV
- Utilize VICON system to fly UAV autonomously
- Compare performance of simulation and actual flight

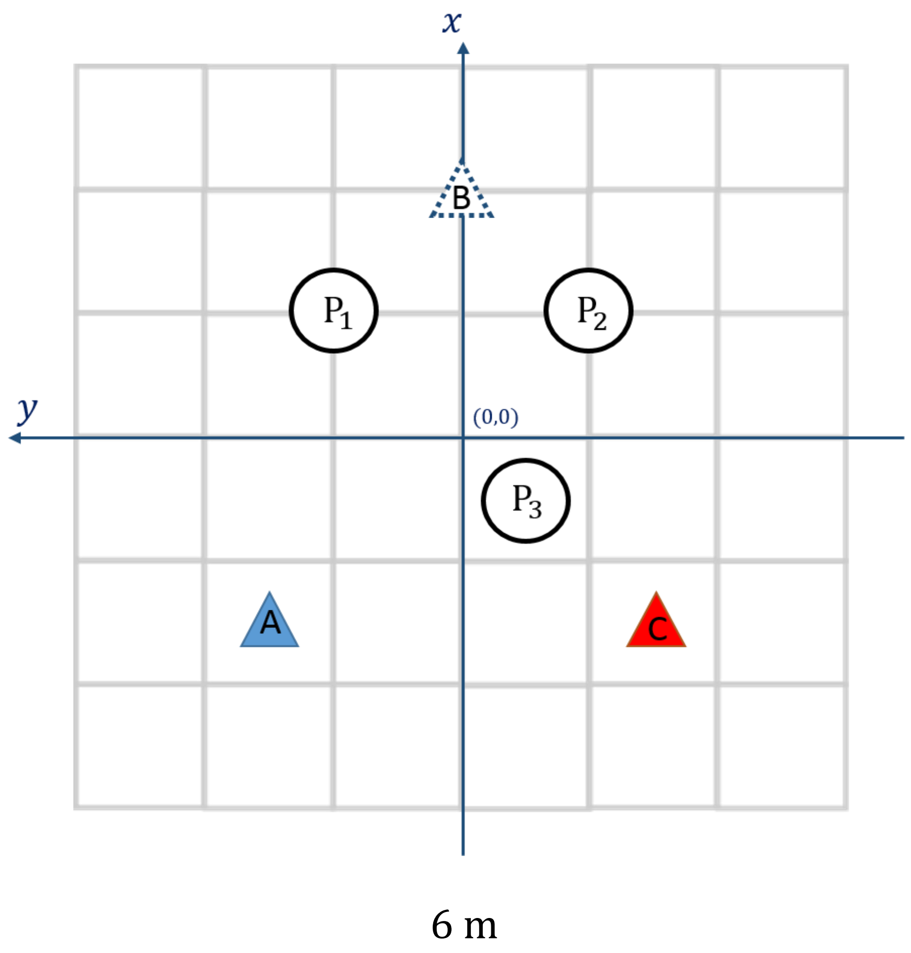
15%

Task 3

- 3 to 5 obstacles will be randomly placed
- Modify simulator of Task 1 to include random obstacles input (if needed)
- Implement and run the design path with the actual UAV in VICON system

10%

Project overview



Path

1. Take-off at point A to 1 meter height
2. Navigate to point B then C while avoiding obstacles
3. Land at point C

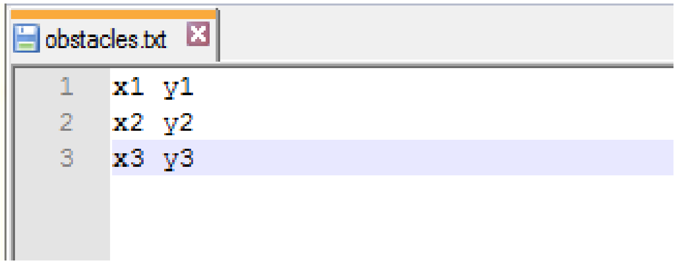
Requirements

1. Center of UAV to be at least 60 cm away from the center of obstacles
2. Design a few different paths to show that the UAV can
   - ❑ Stop at point B before going to point C
   - ❑ Flying over point B with velocity 0.3 m/s

Project Task 1

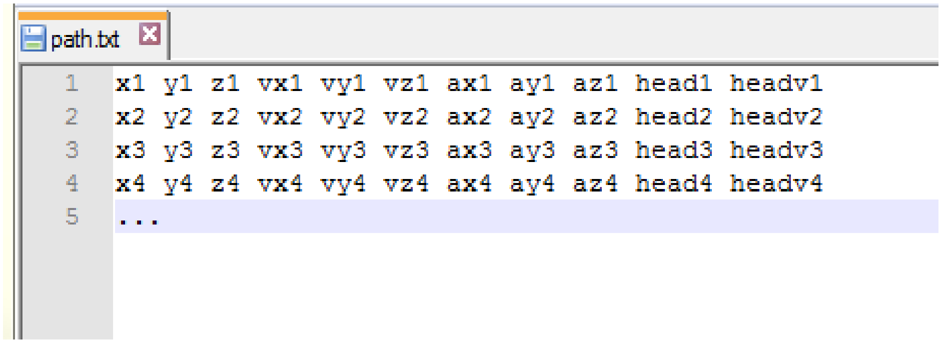
Simulation

1. A simple ROS package for the simulation will be provided, you are encouraged to improve the simulator
2. Simulator inputs 2 files: *obstacles.txt* and *path.txt*
3. *Obstacles.txt* will be given, you are required to generate *path.txt* from your own path-planning algorithm
4. Path should be sampled at 20 Hz (50 ms between data)
5. All variables should be in unit SI, and 3 decimal points.



obstacles.txt

```
x1 y1
x2 y2
x3 y3
```



path.txt

```
x1 y1 z1 vx1 vy1 vz1 ax1 ay1 az1 head1 headv1
x2 y2 z2 vx2 vy2 vz2 ax2 ay2 az2 head2 headv2
x3 y3 z3 vx3 vy3 vz3 ax3 ay3 az3 head3 headv3
x4 y4 z4 vx4 vy4 vz4 ax4 ay4 az4 head4 headv4
...
```

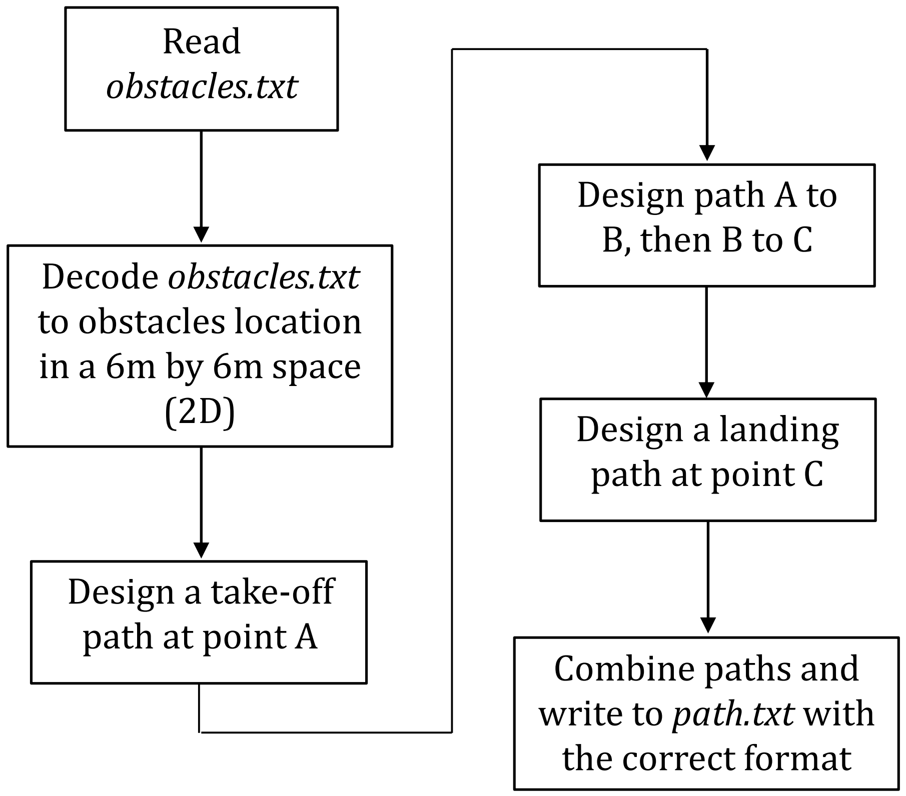
Project Task 1

How to generate *path.txt*

1. Write algorithm in either MATLAB or ROS (C++)

How to write a simulator

1. Write in ROS (C++) only
2. A sample simulator software in Gazebo environment is provided, you are required to modify the simulator for Task 3.


Read *obstacles.txt*
Decode *obstacles.txt* to obstacles location in a 6m by 6m space (2D)
Design a take-off path at point A
Design path A to B, then B to C
Design a landing path at point C
Combine paths and write to *path.txt* with the correct format

Project Task 1

How to install the simulator?

1. Create a workspace

```
$ source /opt/ros/indigo/setup.bash
$ mkdir -p ~/uavWs
// Now copy the simulation source files into the folder
$ cd ~/uavWs/src
$ catkin_init_workspace
```

2. Compile the project

```
$ cd ~/uavWs
$ sudo chmod 777 -R ./
// Now enter your password
$ catkin_make
```

3. Start simulation environment

```
$ ./setup
```

Project Task 1

How to run the simulator?

1. Prepare your *obstacles.txt* and *path.txt*

2. Go to uavWs workspace

```
$ cd ~/uavWs
```

3. Open the simulation script "sim"

4. Substitute the address of your reference file and obstacle file

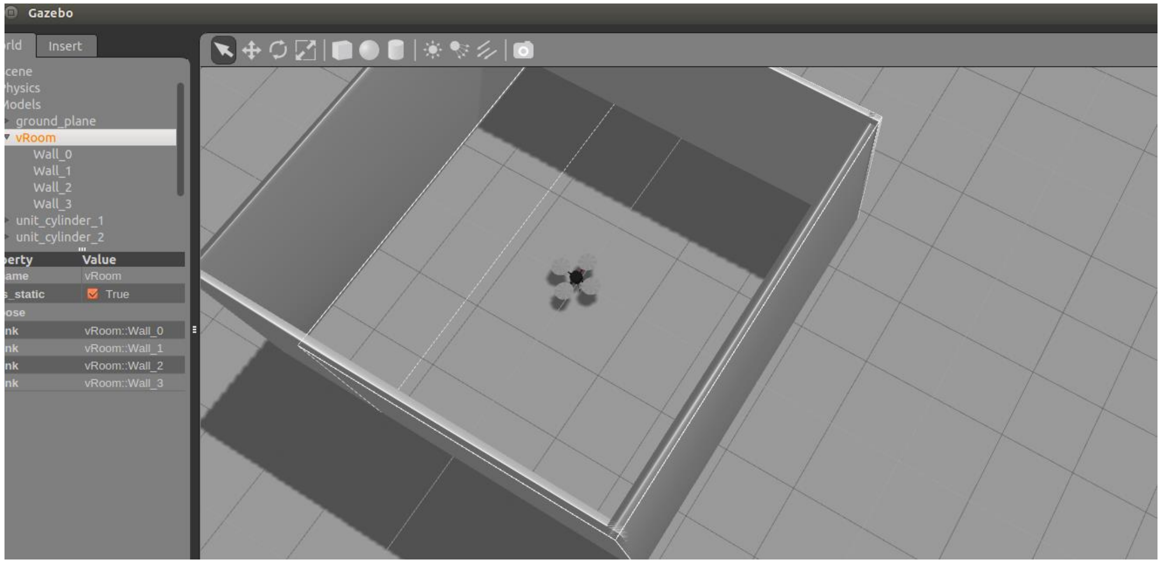
```
$ rosrun hector_quadrotor_reference refPub _obsAdr:="path_of_obstacles.txt"
              _refAdr:="path_of_path.txt"
```

5. Run the simulation by

```
$ ./sim
```

Project Task 1

How to run the simulator?



erty	Value
ame	vRoom
_static	True
ose	
nk	vRoom::Wall_0
nk	vRoom::Wall_1
nk	vRoom::Wall_2
nk	vRoom::Wall_3

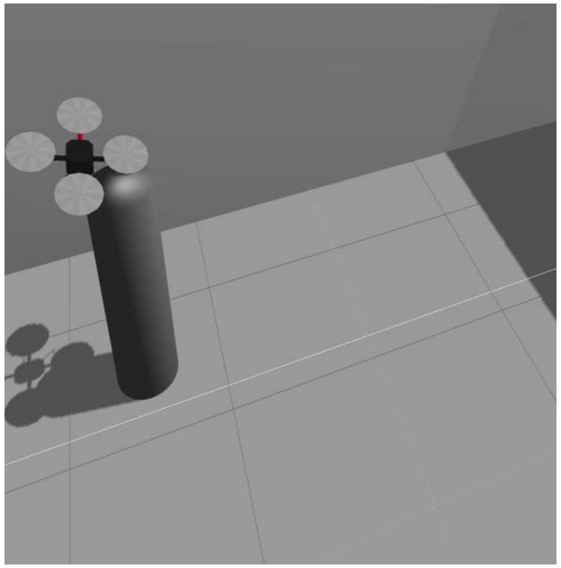
Simulation world

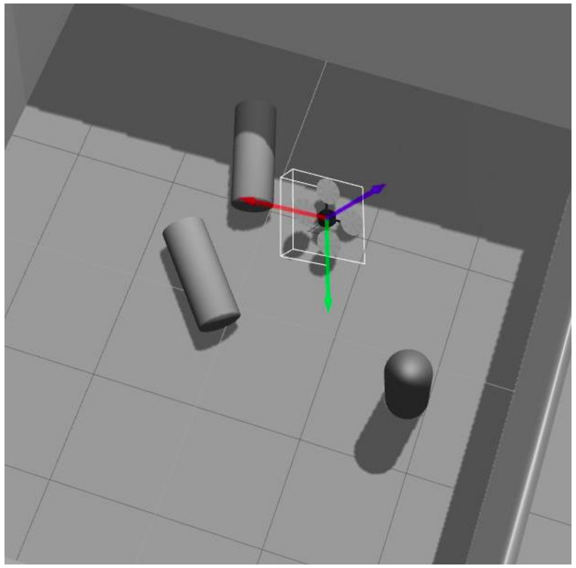
The simulation world is a simple 5.5 m by 5.5 m room. You can change the camera position on your computer freely.

The room size matches the final actual flying area in VICON room. The origin is set at the center of the room. In this simulator, the obstacles and UAV are both slightly bigger than actual object, to account for uncertainty and disturbance in actual experiment.

Project Task 1

How to run the simulator?





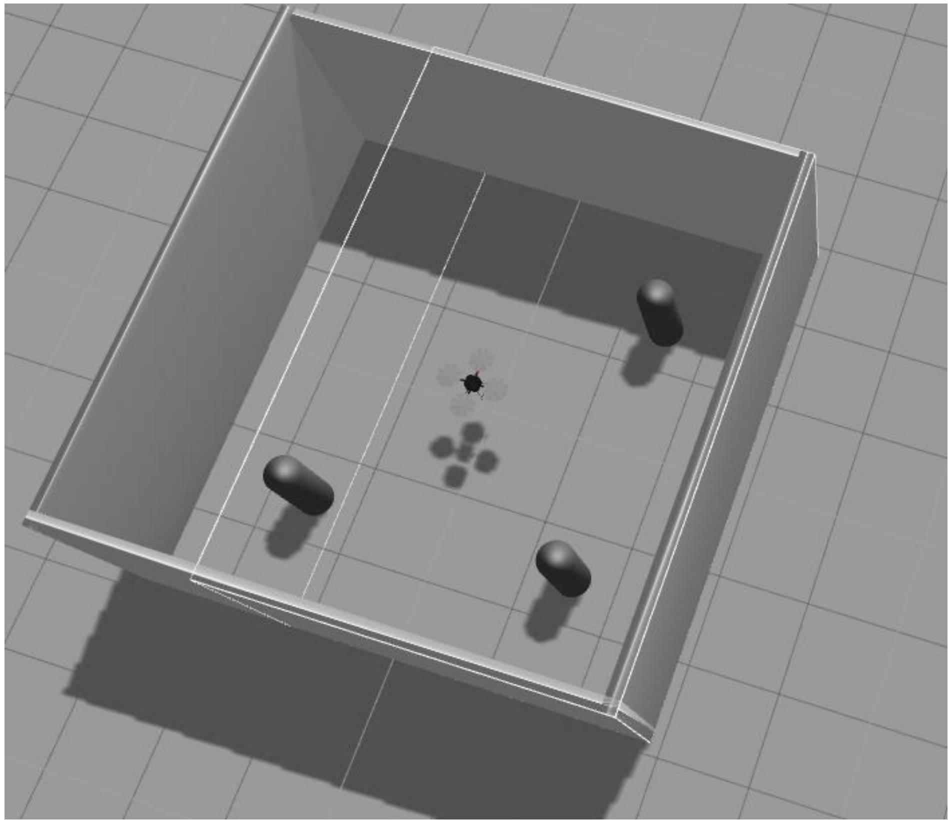
Start simulation

By running the “sim” script, the obstacles (*unit_cylinders*) will be positioned at the corresponding place in the flying zone following your *obstacles.txt* description.

The UAV will be positioned at the first position of your *path.txt* file, then start to track your designed path. If collision occurs, cylindrical obstacles will fall in the display window.

Project Task 1

How to run the simulator?



<u>Restart simulation</u>
If the simulation ended with a collision, you will have to redesign your reference trajectory and run the simulation again.

The simulator in Gazebo environment needs not to turn off. Simply run `$ ./sim` again with a updated path.txt, and it will restart the simulation automatically. Your new trajectory will be shown in the visual display window.

Project Task 1

Additional functions

1. To request the state of the UAV and obstacles:

   UAV

   ```
   $ rosservice call gazebo/get_model_state '{model_name:quadrotor}'
   ```

   Obstacles

   ```
   $ rosservice call gazebo/get_model_state '{model_name:unit_cylinder_1}'
   $ rosservice call gazebo/get_model_state '{model_name:unit_cylinder_2}'
   $ rosservice call gazebo/get_model_state '{model_name:unit_cylinder_3}'
   ```

2. To start and stop the engine of the quadrotor:

   Start

   ```
   $ rosservice call engage
   ```

   Stop

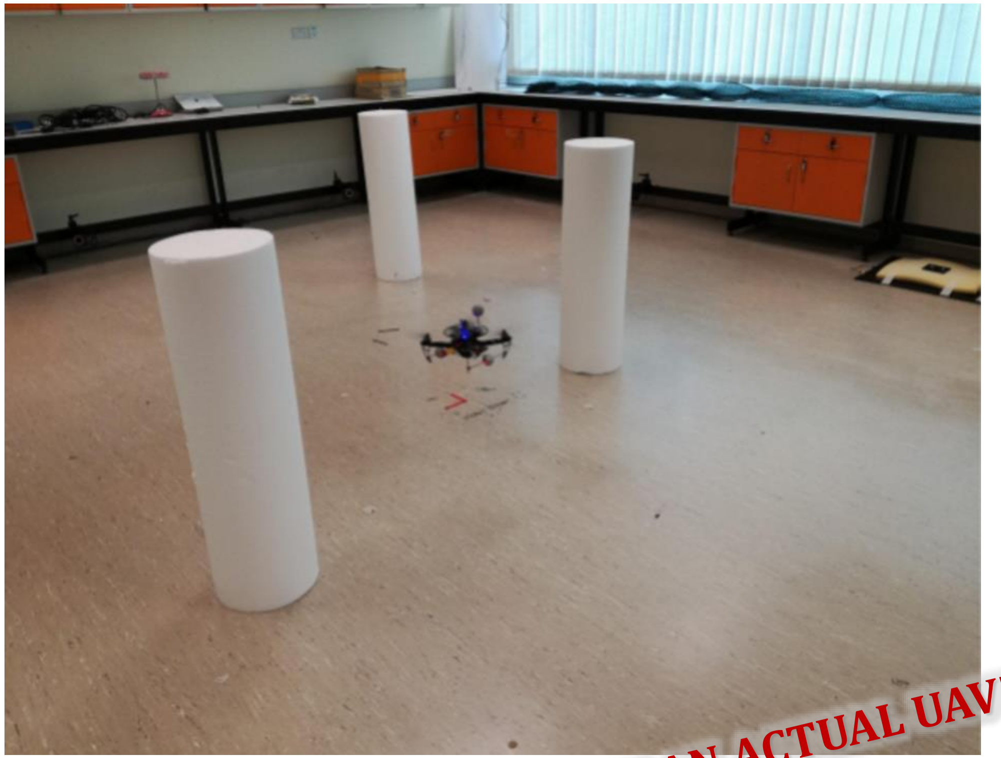
   ```
   $ rosservice call shutdown
   ```

Project Task 1

Before proceeding to Task 2, make sure ...

- ✓ You have completed Task 1 which satisfied all requirement, and without collision with the obstacles
- ✓ Generated at least two *path.txt* files verified in Task 1
- ✓ To bring your simulator source files (e.g: with a thumbdrive) to VICON room for verification

Project Task 2



FLY AN ACTUAL UAV!!!

Path

1. The same path (or multiple paths) which was proven in Task 1
2. Check with TA on duty before proceeding to Task 2 (safety first!)

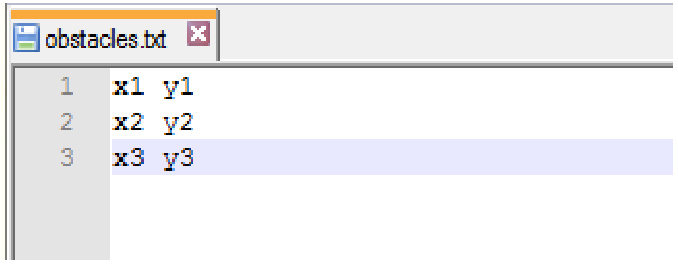
Requirements

1. Same requirements as Task 1
2. Runs at least 2 different paths to show
   - ❑ UAV to stop and go at point B
   - ❑ UAV to fly over point B at 0.3 m/s

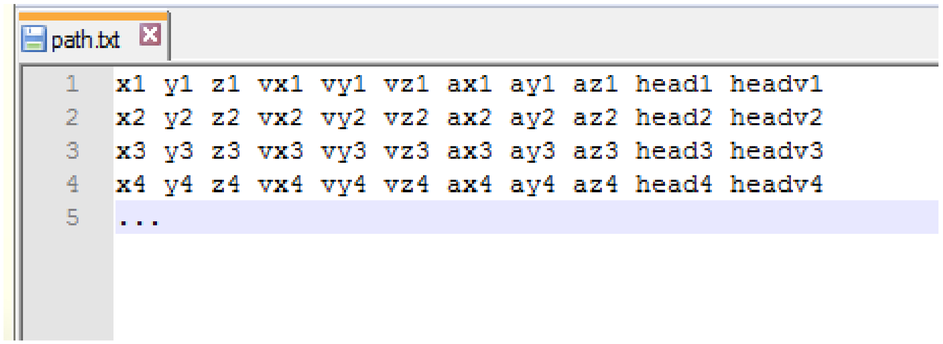
Project Task 3

Advanced missions

1. 3 obstacles will be placed by TA
2. *Obstacles.txt* will be generated on the spot and given to you
3. You have to plan a path with new obstacles location, and generate a new *path.txt*
4. Include the new *obstacles.txt* and *path.txt* into simulator of Task 1
5. If successful in simulator, implement to actual UAV
6. Repeat experiment with 4 or 5 obstacles


obstacles.txt

```
x1 y1
x2 y2
x3 y3
```


path.txt

```
x1 y1 z1 vx1 vy1 vz1 ax1 ay1 az1 head1 headv1
x2 y2 z2 vx2 vy2 vz2 ax2 ay2 az2 head2 headv2
x3 y3 z3 vx3 vy3 vz3 ax3 ay3 az3 head3 headv3
x4 y4 z4 vx4 vy4 vz4 ax4 ay4 az4 head4 headv4
...
```

Note: ***To complete this challenge, your algorithm and simulator should take into account of any number (between 3 to 5) of obstacles, and at any location within the flying zone.***

Path planning algorithm

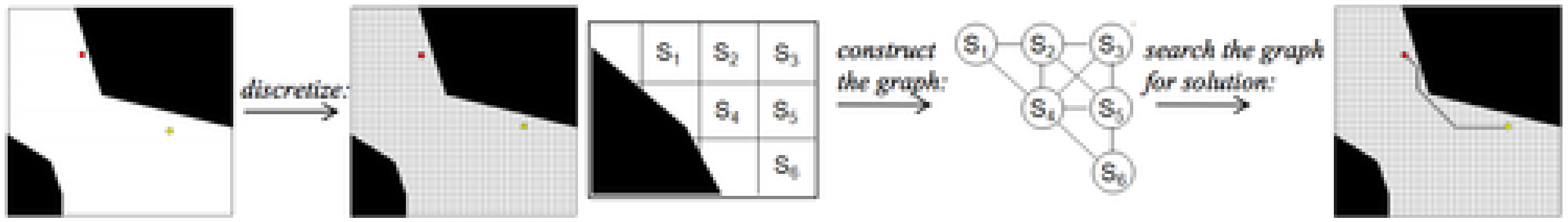
2D or 3D?

UAV can fly in any 3 directions (front/back, left/right, up/down), and thus intuitive way to design the path will be in 3D space.

However, for simplicity, in this project you can design a 3D path in a 2D/1D manner by breaking down the problem

1. Take-off (only vertical direction, 1D)
2. Navigation (only horizontal direction, 2D)
3. Landing (only vertical direction, 1D)

Path planning algorithm



World discretization

In this project, we focus on the search based planning. The first step is to discretize the world (not only the configuration space but also the state space), such that we can get a connected graph.

Graph searching technique can be used once we get a connected graph.

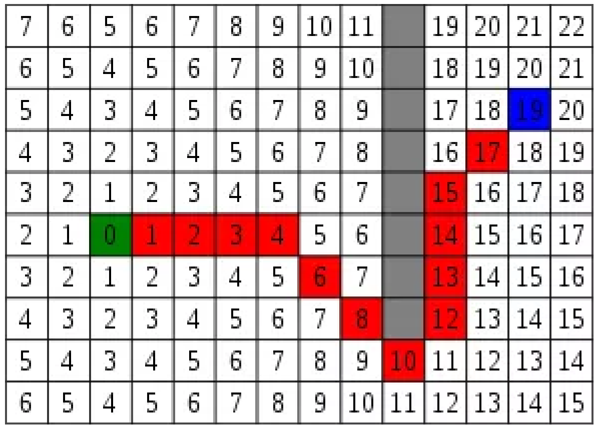
Path planning algorithm

UAV as omni-directional robot

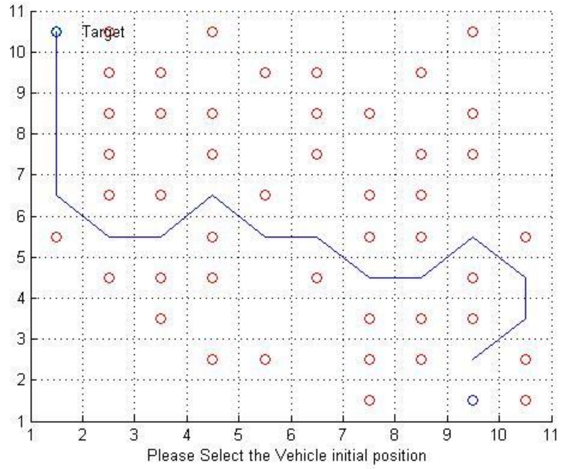
UAV in quadrotor form can fly in any direction, it can be treated like an omni-directional robot.

At low speed, it is safe to assume the robot can travel at any direction. Therefore, it is reasonable to discretize its world based on a 2D surface.

The robot can travel to one of its 8 neighbours at any moment.



7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15



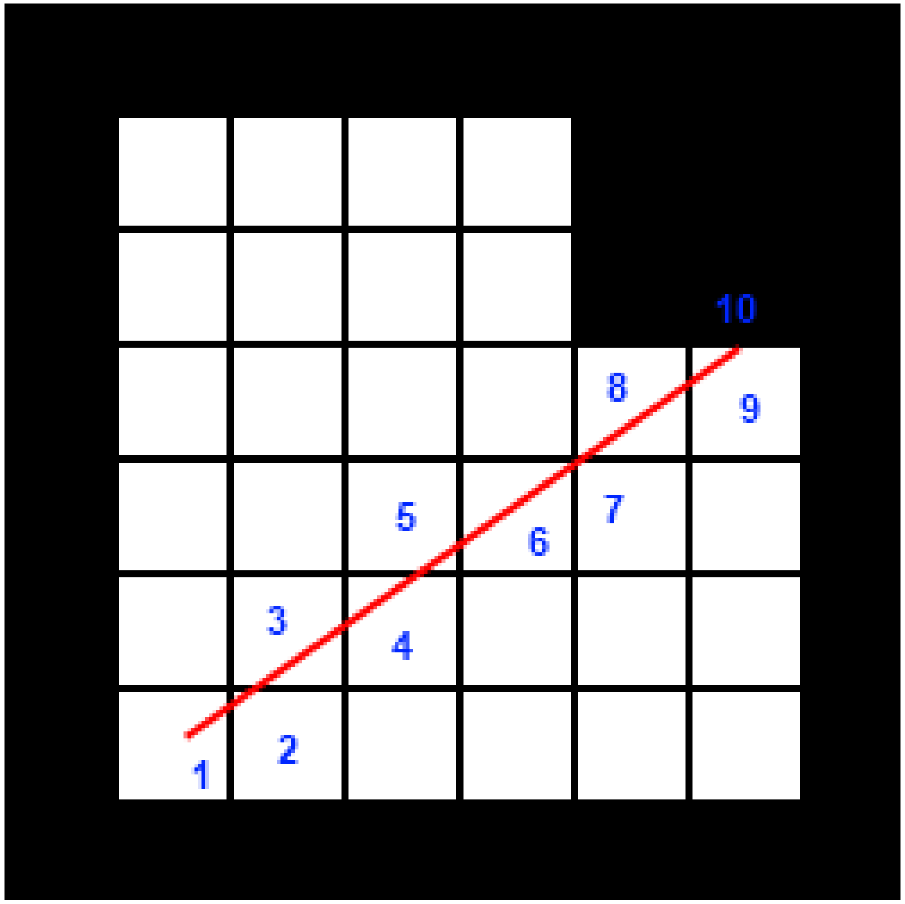
Path planning algorithm

Collision checking

One of the important aspects in path planning is the collision checking.

For the discretized grid world, the ray-casting based collision checking technique can be used.

The basic idea is to find all the grids covered by the path of the robot, then check if all the grids are free of collision (occupied or empty)

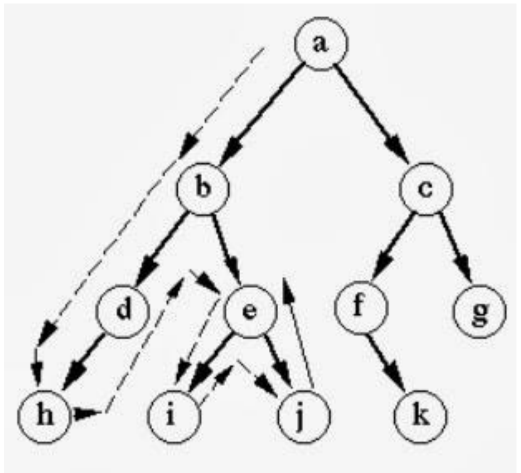


Path planning algorithm

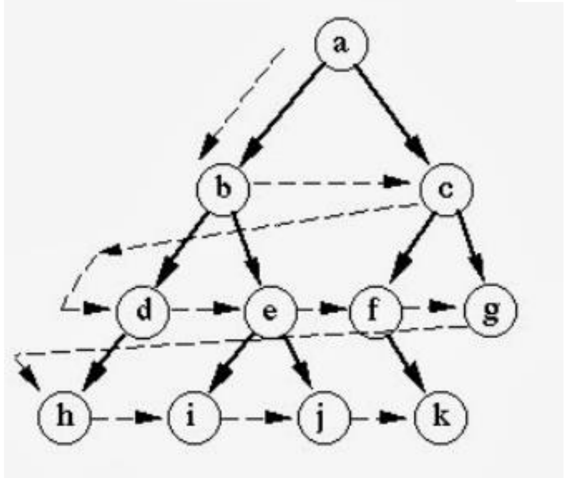
Search algorithm
One of the most fundamental method in path planning is the search algorithm. Each available neighbor in the discretized map can be treated as node, and each node will connect to subsequence nodes in the same manner.

Search algorithms can be classified based on their mechanism of searching, or in other words, based on optimizing the 'best' node. By defining different type of 'best', there are a few popular search algorithms:

- ❑ A* algorithm
- ❑ Dijkstra's algorithm
- ❑ Greedy algorithm



Depth-first search

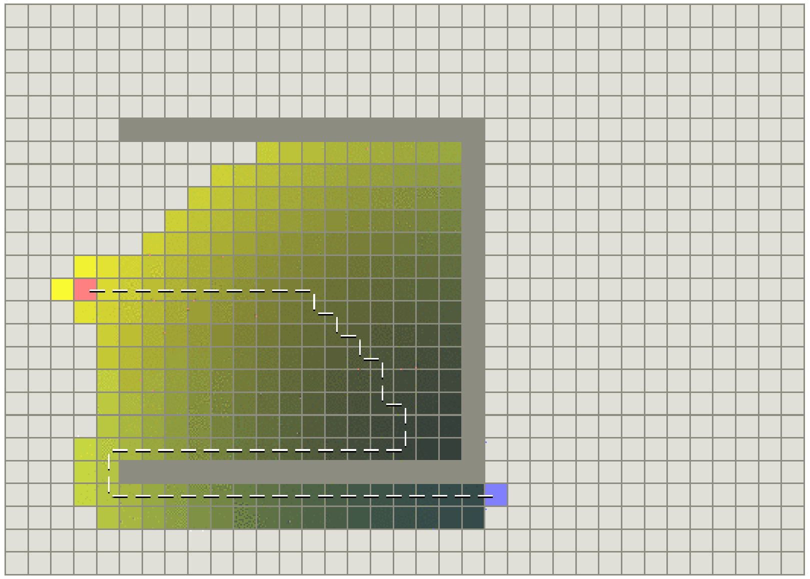


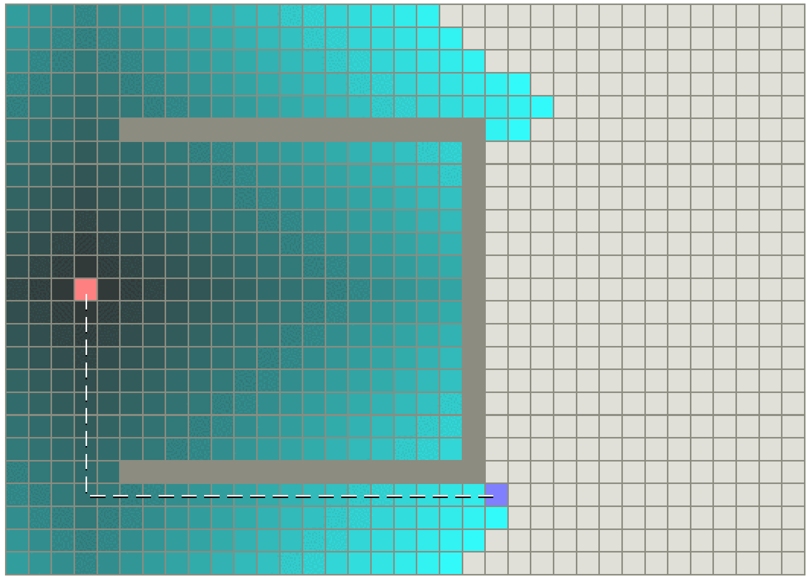
Breadth-first search

Path planning algorithm

Greedy v.s. Dijkstra

The difference between Greedy search and Dijkstra search is on the definition of ‘best’ node. Greedy search always choose the node closest to the final target, while Dijkstra search always choose the node closest to the start target.





Path planning algorithm

Greedy

Advantages:

- ➢ Easier to implement
- ➢ Require less computing power
- ➢ It runs faster

Disadvantages:

- ➢ Might not reach global optimum

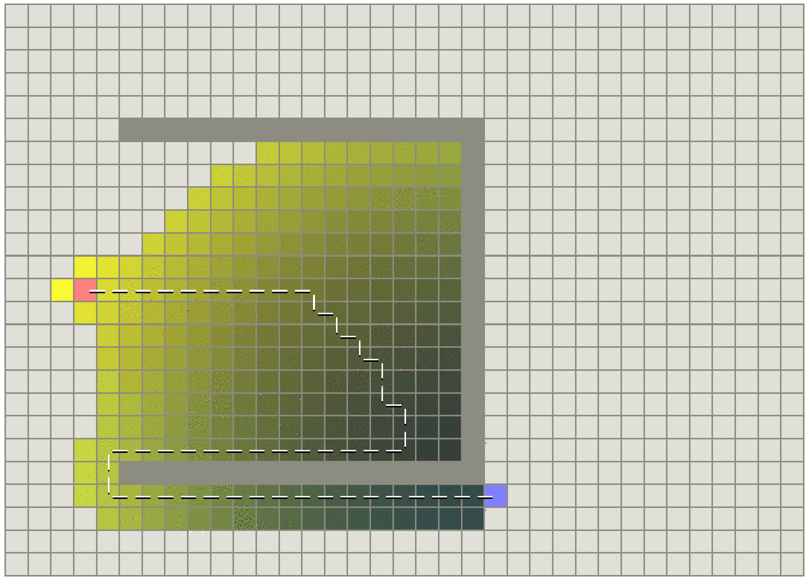
Dijkstra

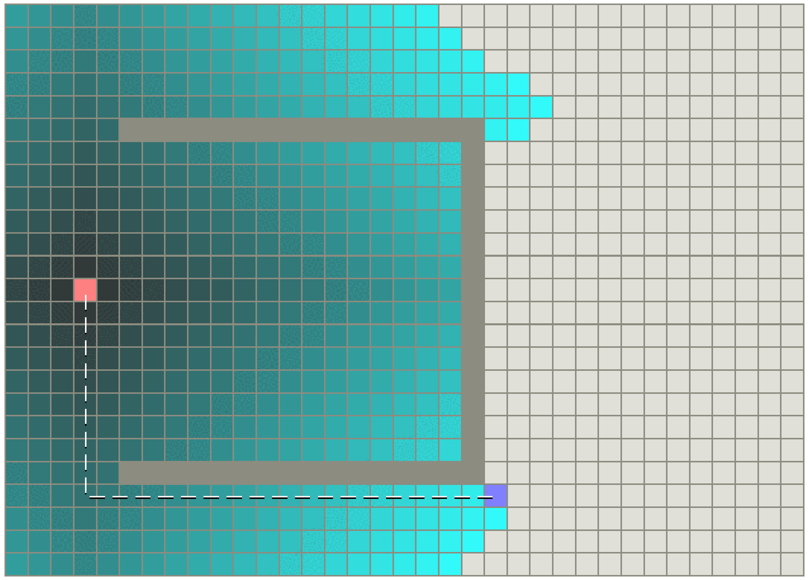
Advantages:

- ➢ Global optimum guaranteed

Disadvantages:

- ➢ Run time longer



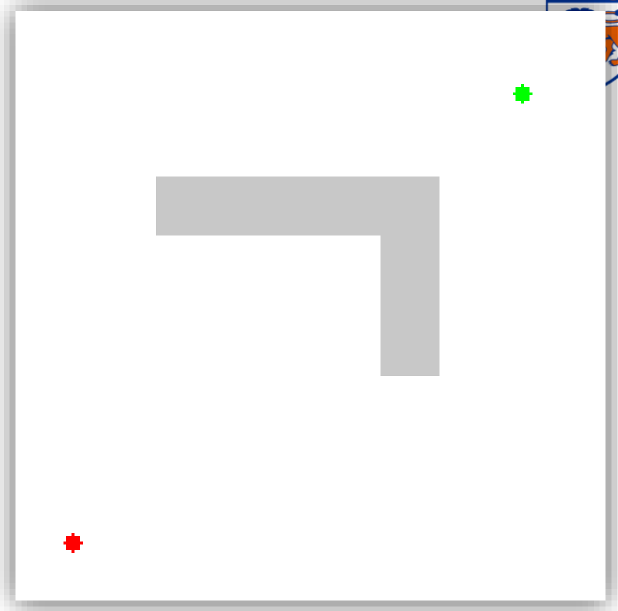


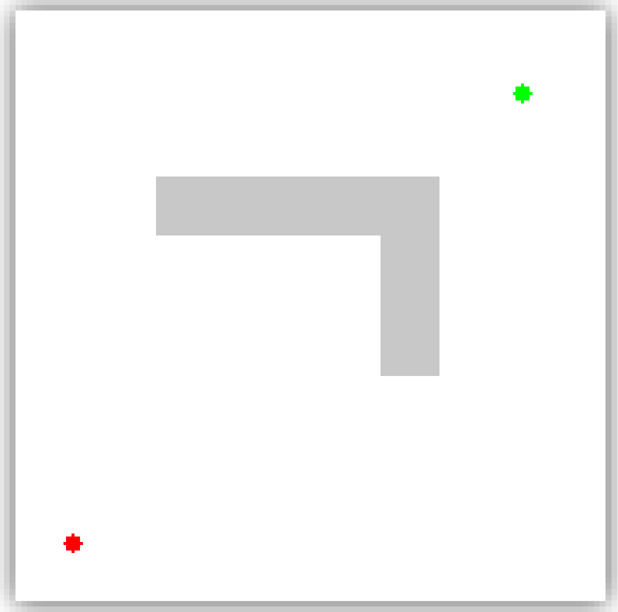
Path planning algorithm

A* search algorithm
A* solves problems by searching among all possible paths to the solution for the one that incurs the smallest cost (least distance travelled), and among these paths it returns the ones that appear to lead most quickly to the solution.

Advantages:
- It has sub-optimum solution
- Faster than Dijkstra

Disadvantages:
- Certain cases cannot find optimum path
- Algorithm more complex



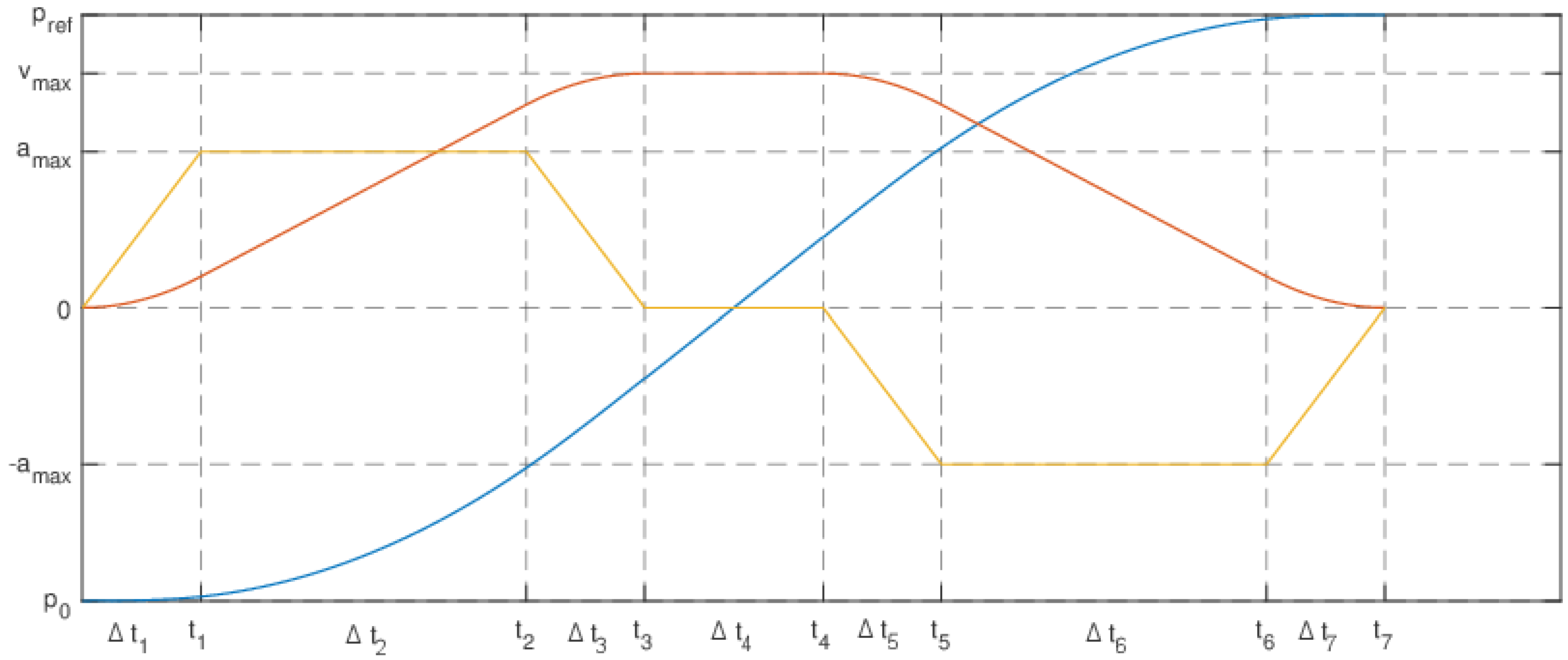


Path planning algorithm

Velocity profile

It is not a good idea to give step reference to the robots, as it will cause large control input to the robot and practically impossible changes on the motor speed.

A smooth trajectory with continuous velocity, acceleration (and sometimes jerk) needs to be considered in planning path for robots.



Path planning algorithm

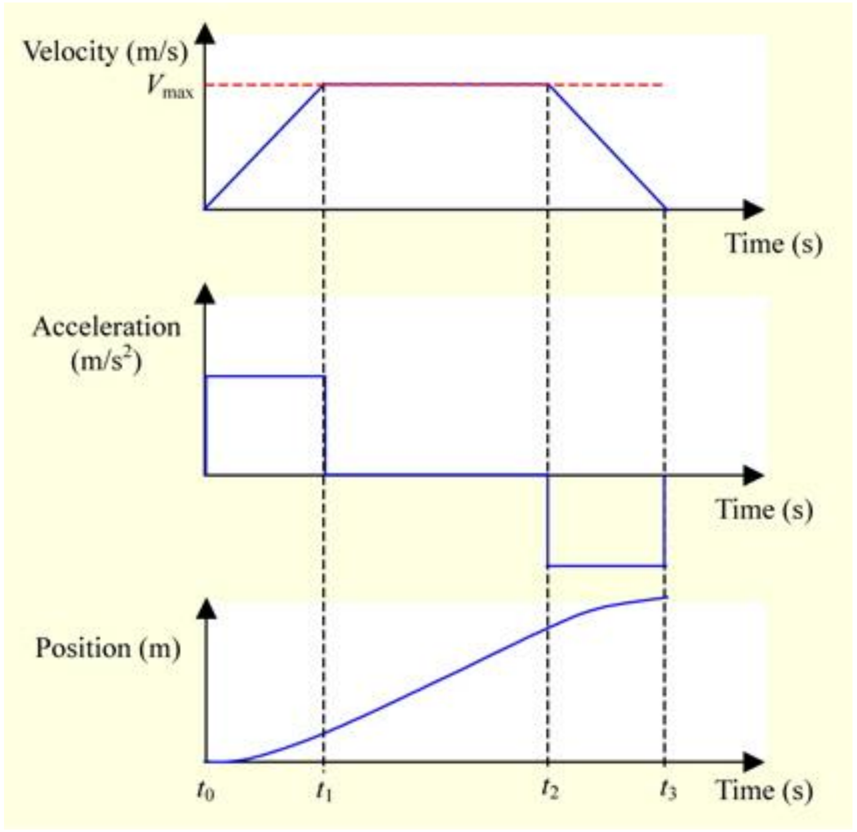
Velocity profile

Example: A trapezoidal velocity profile

You need to make sure:

1. The desired distance is correctly covered in this time period

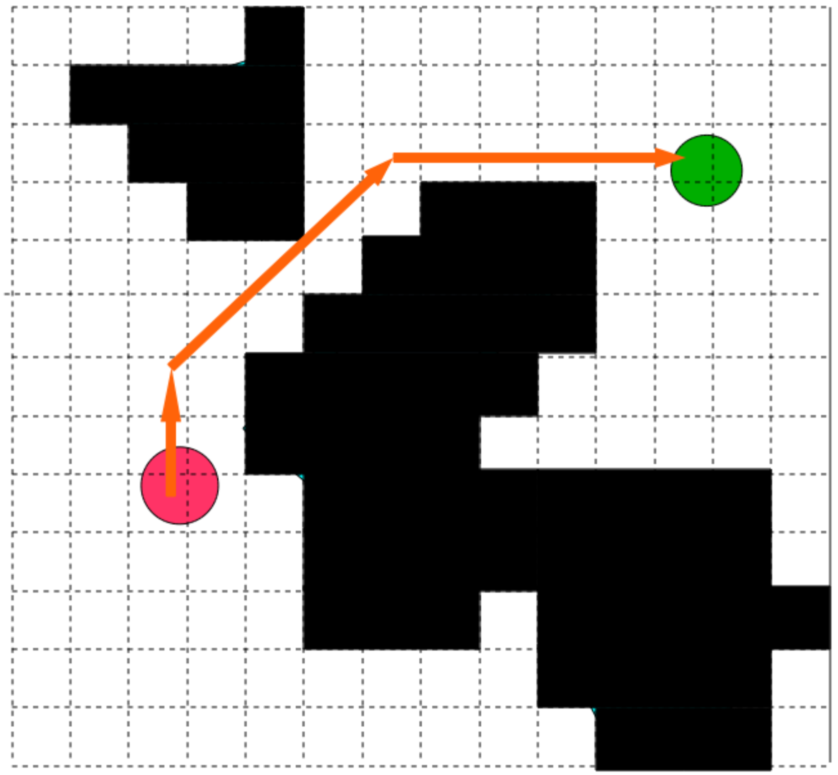
2. The vehicle stopped at the desired point as overshoot might leads to collision.



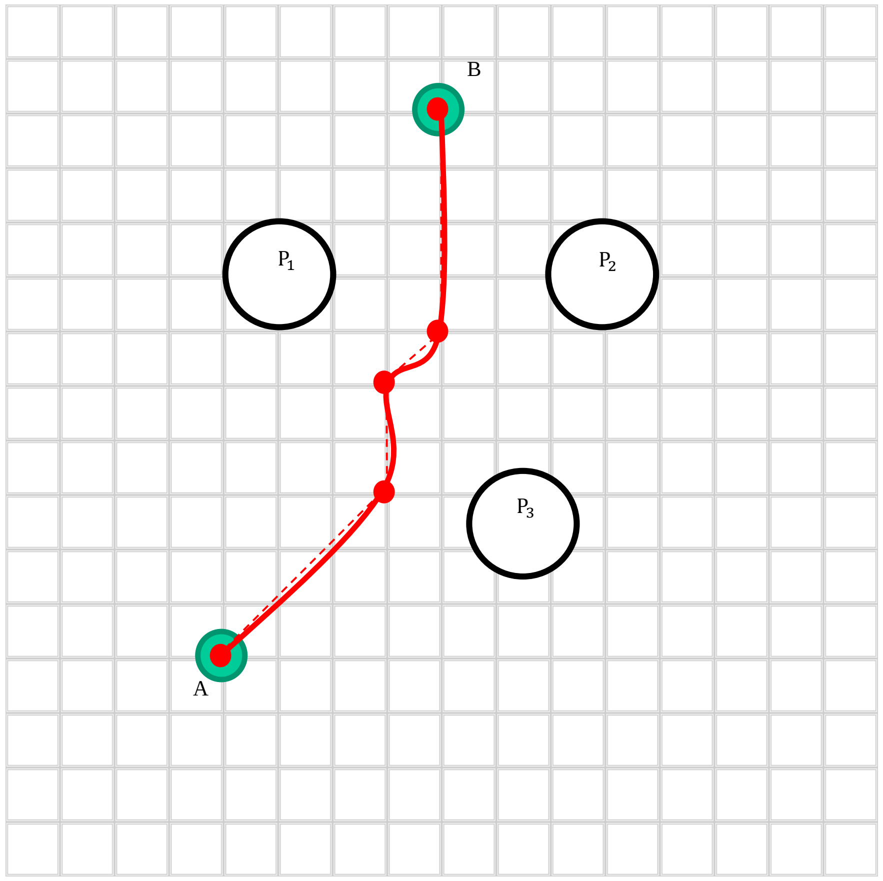
Path planning algorithm

Overall strategy

1. Examine the environment clearly to choose a best path planning method
2. Discretize the world with reasonable grid size
3. Project the vehicle and obstacle (or the map) into the grid world
4. Run searching algorithm to generate line segment path from start to end
5. Generate velocity profile on each line segment



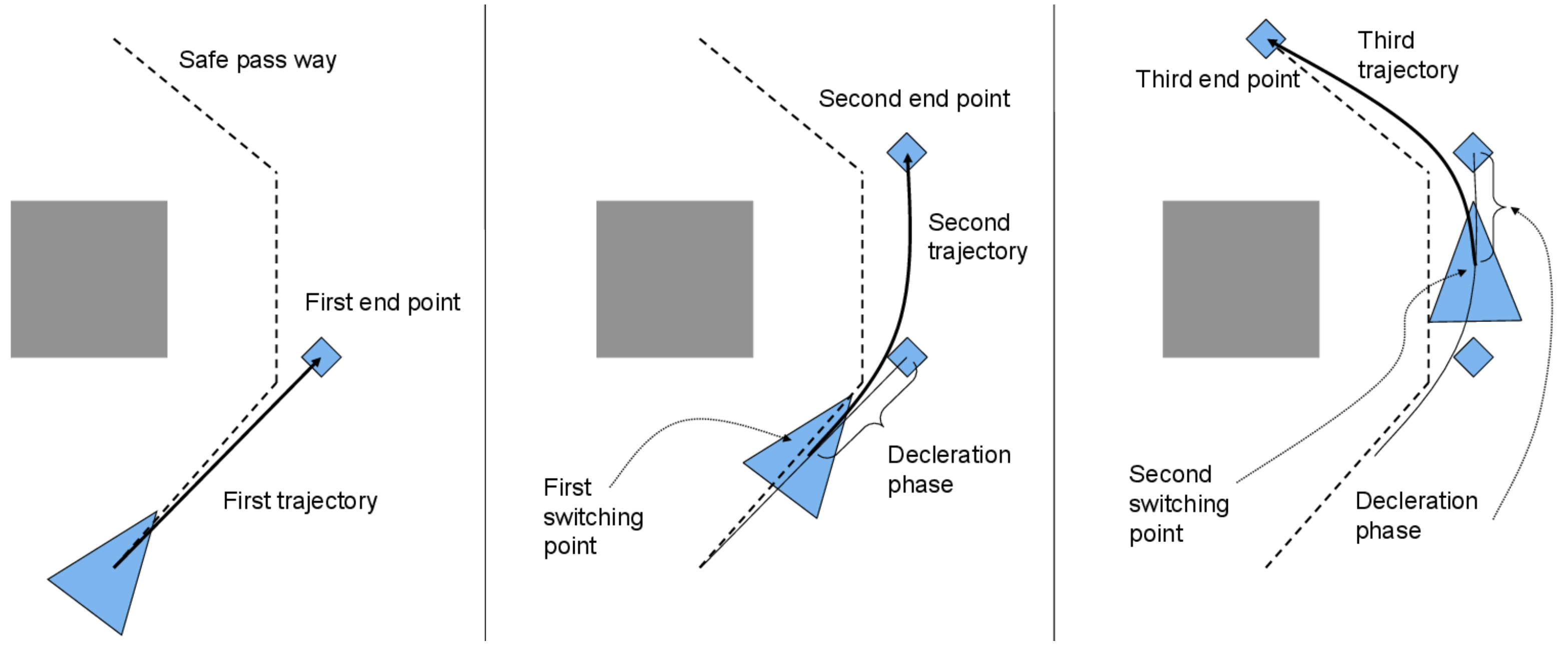
Path planning algorithm



Overall strategy

1. Observe the map and find a suitable grid to discretize
2. Use search algorithm to find path to point B
3. Apply velocity profile to connect each nodes
4. Combine trajectory between each nodes to a complete path.

Path planning algorithm



Possible improvements

1. Line segments are generated, vehicle flying towards the end of first segment
2. Before reaching the end, it switch to travelling to the next line segment
3. Repeat till it reach the final end point.

Difficulty here is to maintain the continuity of the reference signal, and initial velocity of each trajectory segment is not zero!!

Reminder ...

1. Register for your group in IVLE

2. Choose suitable dates for your Task 1 simulation in M&A Lab using ROS desktop

3. Try to modify Task 1 simulator to include 4 and 5 obstacles

4. Choose suitable dates for Task 2 flight experiment (email skphang@nus.edu.sg)

Questions?

Welcome to visit our group website at

http://uav.ece.nus.edu.sg

for more information on our research activities and published resources…