**SLAM Trainee Module**

## **Instructions**

* Parts of the module highlighted in green are “checkpoints” and trainees are required to update the following task sheet:

[Add your name, doc link, GitHub Link](https://docs.google.com/spreadsheets/d/1P7vThV58AyF2Hugs6Z4gt2Bge9BDLyrdGIdA61ZMiVc/edit?usp=sharing)

* Documentation is a must for every checkpoint and trainees are required to create a Google Doc sheet where they document their learning, errors encountered and doubts. This google doc must be made accessible to “anyone with link” and the link for the same must be updated on the task sheet.

~~Feel free~~ you are **encouraged** to ask doubts to JDEs if you feel stuck or want to understand a topic better

* Keep in mind, performance in modules is how the team will judge your abilities and effort to assign subsystems once the time comes
* Have a fun learning experience!

## Competition Overview:

Our team focuses on participating in Formula Student UK/Germany/Netherlands etc and these competitions have their own specific problem statements. In general, our race car is required to navigate through a racetrack defined by cones on either side (yellow cones on the left and blue cones on the right). We are not given the map of the track, so we need to create one ourselves, that’s where slam comes in. Along with drawing the track's layout, Slam is also in charge of localizing (where the car is) the car on the given map.

## Flow of the module:

* Understanding SLAM and why it is needed?
* Basics of ROS: command-line ros tools, rosbags, rviz
* Use of Kalman filters to solve Localization
* Checkpoint 1
* Moving to Extended Kalman Filters use in SLAM
* Motion Update
* Measurement Update
* Checkpoint 2
* Subsystem Overview and progress made by us so far
* Data Association
* Checkpoint 3

# **WHAT IS SLAM AND WHY DO WE NEED IT?**

Watch [Introduction to SLAM (Cyrill Stachniss)](https://youtu.be/0I30M6yTklo?si=phe4AZwgFEISCtii) to better understand the text given below 🙂

There are many occasions when an autonomous system like a robot needs to navigate in an environment that is mostly unknown to it beforehand. **Navigation** involves determining where the robot itself is in the environment **(localization)** as well as determining where and how it must go to its destination **(path planning)**. It seems that the aspect of localization becomes impossible without having a map or model of the environment. Thus, it becomes necessary to create a map of the environment while the robot navigates through the environment.

SLAM problem is hard because it is kind of a paradox i.e :

* In order to build a map, we now need the position.
* To determine our position, we need a map.

It is like a chicken-egg problem. In practice these two problems can be solved independently of each other. A good map is needed for localisation while an accurate pose estimate is needed to build a map.

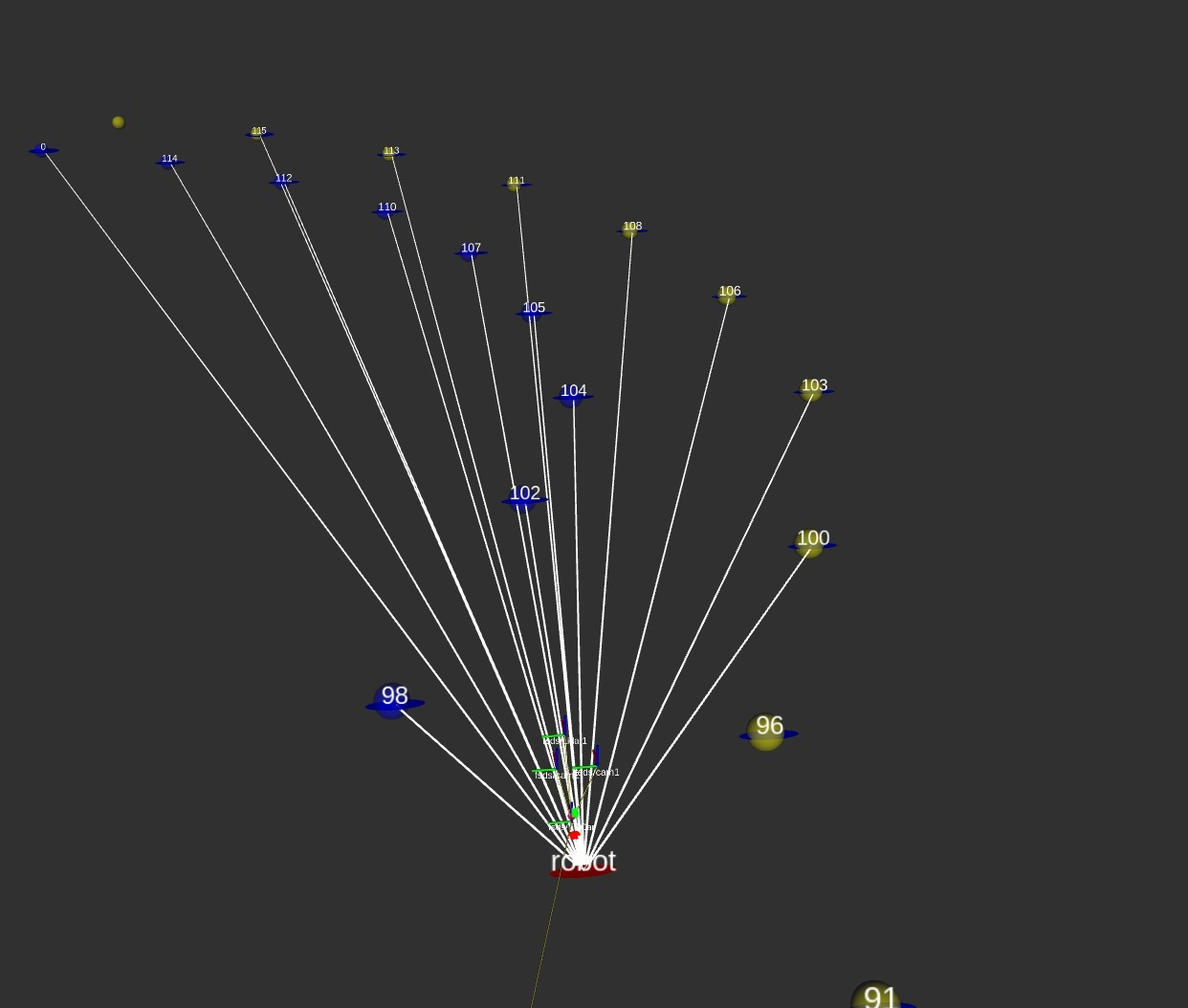
However, techniques have been developed that enable a robot to do localization and mapping efficiently at the same time. These techniques come under the umbrella of **Simultaneous Localization and Mapping** or **SLAM** in short.

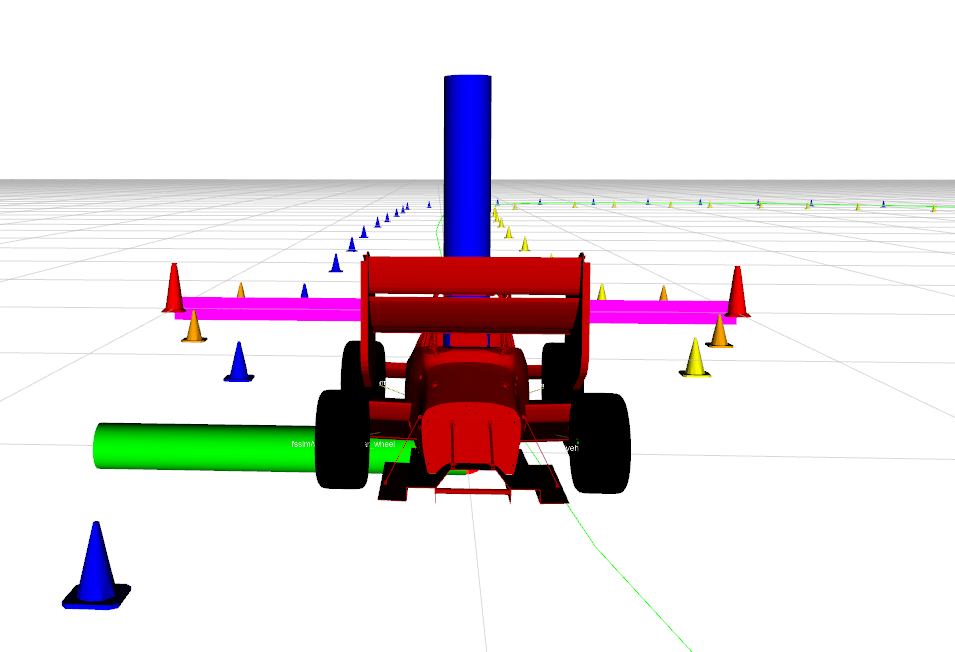
no

*Car's trajectory between blue and yellow cones using MRPT and PPC algorithms. White lines show cones detected, while blue and red ellipses represent covariance in positions.*

The figure below is a visualization of an autonomous race car on a typical track. As you can see in this figure, the track boundaries consist of blue ones on the left and yellow cones on the right. The orange cones designate the start of the track and they play a role in determining whether the car has completed a lap. The large reddish cones are the time-keeping devices that are used to measure the lap times.

Consider the scenario in which this autonomous race car is at rest at the start position on a track it has never been to, so it has no idea how to move around at first. In other words, the car does not have a map of the track that it can rely on to know where it is and how it must move



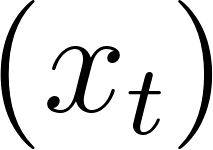
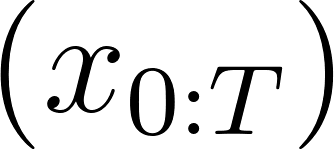


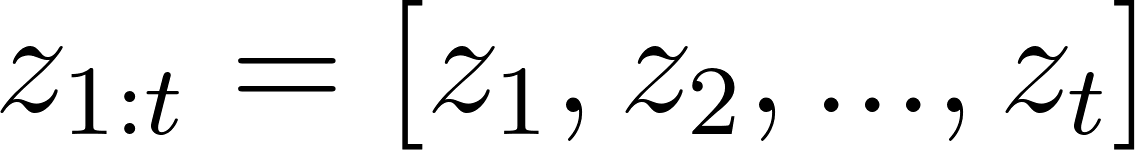
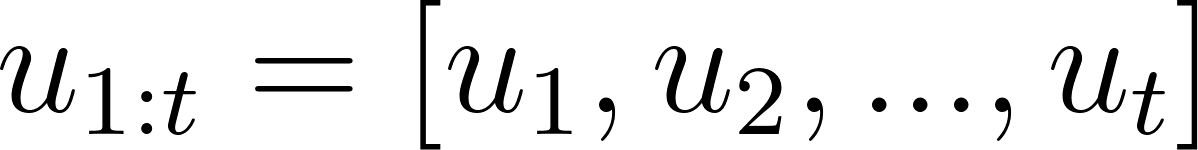
ahead. Thus the task of creating a map of the track is important. But it is not realistically possible for the car to make a map by using cameras or LiDAR by staying in one place. Thus it must move around as well and use the knowledge it has of how it moved and what it detects using its sensors to construct a map of the track and determine its position on the track. This does sound difficult and this is the challenge we are dealing with!

[Autonomous Racing: AMZ Driverless with flüela](https://youtu.be/FbKLE7uar9Y)

Observe this [video](https://youtu.be/FbKLE7uar9Y), where the car performs SLAM in the first lap, and once it makes the map, switches to just localization in later laps & car go brrrr.

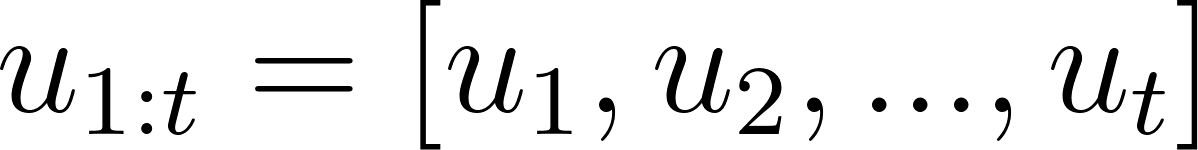
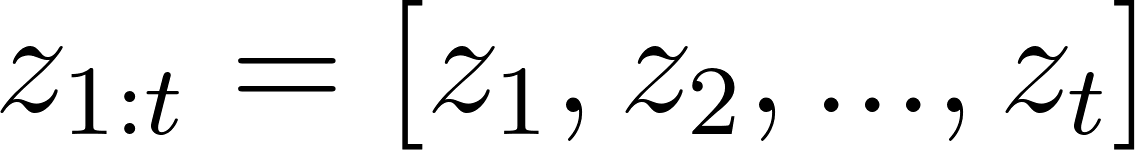
# **The formal problem statement**

Given the set of controls at different time steps [](https://www.codecogs.com/eqnedit.php?latex=%20(%20u_t%20)%20#0) and a set of observations [](https://www.codecogs.com/eqnedit.php?latex=%20(z_t)%20#0) over time, the problem is to estimate the current state of the robot [](https://www.codecogs.com/eqnedit.php?latex=%20(x_t)%20#0)/ entire path of the robot [](https://www.codecogs.com/eqnedit.php?latex=%20(x_%7B0%3AT%7D)%20#0) and the map of the environment [](https://www.codecogs.com/eqnedit.php?latex=%20m%20#0). This is done in a probabilistic manner due to the existence of noise and bias in the controls as well as sensor observations.

The set of observations and set of control commands till time t are collectively represented as [](https://www.codecogs.com/eqnedit.php?latex=%20z_%7B1%3At%7D%20%3D%20%5Bz_1%2C%20z_2%2C...%2C%20z_t%5D%20#0) and [](https://www.codecogs.com/eqnedit.php?latex=%20u_%7B1%3At%7D%3D%5Bu_1%2Cu_2%2C...%2Cu_t%5D%20#0)

The Simultaneous Localization and Mapping (SLAM) problem is a fundamental task in robotics that involves constructing a map of an unknown environment while simultaneously estimating the robot's pose within that map. This is done in a probabilistic manner due to the existence of noise and bias in the controls as well as sensor observations. It can be mathematically defined as follows:

Given:

* A sequence of robot control inputs: [](https://www.codecogs.com/eqnedit.php?latex=%20u_%7B1%3At%7D%3D%5Bu_1%2Cu_2%2C...%2Cu_t%5D%20#0) where represents the control input at time step t.
* A sequence of robot sensor measurements:[](https://www.codecogs.com/eqnedit.php?latex=%20z_%7B1%3At%7D%20%3D%20%5Bz_1%2C%20z_2%2C...%2C%20z_t%5D%20#0) where represents the sensor measurement at time step t.
* An initial estimate of the robot's pose:
* An initial estimate of the map of the environment:

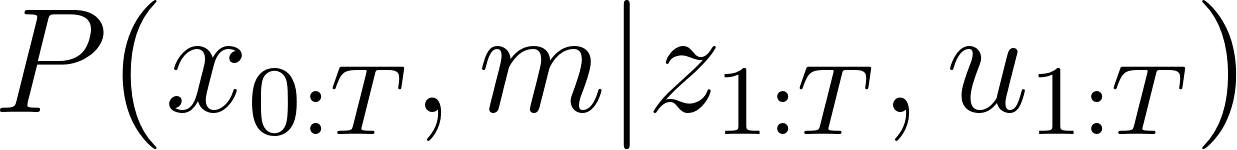
The goal of SLAM is to estimate the robot's trajectory (position as well as the direction where it is heading ) over time -  ***,***  and to build a map of the environment, ***m***, by optimizing the joint posterior distribution:

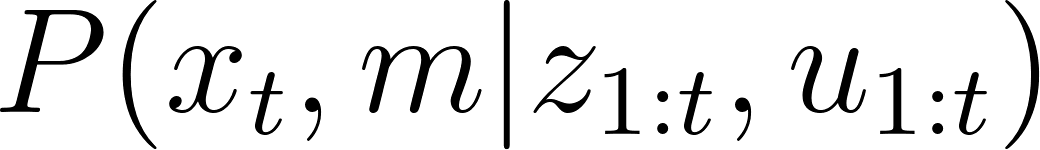
This joint distribution represents the probability of the robot's trajectory and the map given the sensor measurements and control inputs. The SLAM problem involves finding the maximum a posteriori (MAP) estimate of this joint distribution.

To solve the SLAM problem, various techniques and algorithms are used, such as Extended Kalman Filter (EKF) SLAM, FastSLAM, GraphSLAM, or Particle Filter SLAM. These algorithms employ probabilistic approaches, such as the Kalman filter or particle filters, to estimate the robot's pose and update the map based on the sensor measurements and control inputs.

The SLAM problem is challenging because it requires addressing the uncertainties in both the robot's motion and the sensor measurements while simultaneously estimating the robot's pose and constructing an accurate map of the environment. This problem is crucial for tasks such as autonomous navigation, mapping of unknown areas, and long-term localization in robotics.

# **Types of SLAM(how is it operated)**

**Full SLAM** involves estimating the entire path of the robot after all the sensor observations are collected and all control commands till time T are executed. This can be mathematically expressed as determining represents the probability distribution about the robot's pose trajectory(position and direction of movement of the robot) and the map. It takes into account all the sensor measurements and control inputs available up to time step t. This approach is useful when the entire dataset is available and allows for more accurate and globally consistent estimates of the robot's trajectory and the environment.

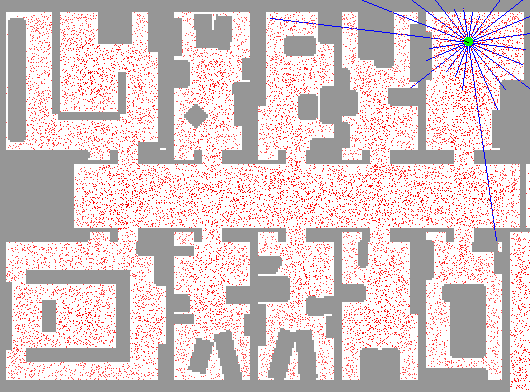
**Online SLAM** involves estimating the current state of the robot based on the sensor observations and control commands till the current time t. This can be expressed mathematically as determining [](https://www.codecogs.com/eqnedit.php?latex=%20P(x_t%2Cm%20%7C%20z_%7B1%3At%7D%2C%20u_%7B1%3At%7D%20)%20#0). In many real-time applications, Online SLAM is more relevant. represents the probability distribution that captures the updated knowledge about the robot's pose at time step t and the map, given all the sensor measurements and control inputs up to time step t. This approach is suitable for real-time applications where the robot needs to continuously update its estimate of the environment while operating.

Which one do you think, would we want to implement in our case? Full or Online?

# **Types of SLAM(which algorithm used)**

**Kalman Filter Based:** Filter-based SLAM algorithms, such as Extended Kalman Filter (EKF) SLAM and its variants, use recursive filtering techniques to estimate the robot's pose and update the map incrementally. These algorithms maintain a belief over the robot's pose and map based on the sensor measurements and control inputs. EKF SLAM is well-suited for environments with linear dynamics and Gaussian noise assumptions, but it can struggle with nonlinearities and large uncertainties.

**Particle Filter Based:** Particle Filter SLAM, also known as Monte Carlo SLAM or Rao-Blackwellized Particle Filter (RBPF) SLAM, combines the benefits of particle filters with the SLAM problem. It uses a set of particles to represent the belief over the robot's pose and map. Particle filters allow non-Gaussian representations and handle nonlinearities effectively. Particle Filter SLAM algorithms use resampling and importance sampling techniques to update the particles based on the sensor measurements and control inputs, estimating the robot's trajectory and constructing the map.



(Laser Scans)

**Graph Based:**  GraphSLAM and its variants, treat the SLAM problem as a nonlinear optimization problem. They formulate the SLAM problem as a graph, where the nodes represent robot poses and landmarks, and the edges represent the constraints between them. By optimizing the graph using techniques like least squares or non-linear optimization methods, these algorithms jointly estimate the robot's trajectory and the map. Graph-based SLAM can handle nonlinearities and non-Gaussian noise better than filter-based approaches.

Watch this [video](https://www.youtube.com/watch?v=L51S2RVu-zc&t=469s) (till 8 min) for a brief history of slam.

Sirf theory se gaadi nahi chalegi

(Just theory won’t help the car move)

Now, let's delve into the beating heart of robotics endeavors—The Robot Operating System (ROS), not actually an operating system but a framework for writing robot software. It is a

collection of tools, libraries, and conventions that aim to simplify the task of creating

complex and robust robot behaviour across a wide variety of robotic platforms.

This pertains to programming robots, not specifically limited to SLAM. So every member of the team is required to have a basic knowledge of ROS, Git stuff.

Your task for now is to get a running version of ROS2 galactic in your Ubuntu systems.

[ROS 2 Documentation: Galactic documentation](https://docs.ros.org/en/galactic/index.html)

Next is to understand the basics of ROS: [Beginner: CLI tools — ROS 2 Documentation: Galactic documentation](https://docs.ros.org/en/galactic/Tutorials/Beginner-CLI-Tools.html), [Writing a simple publisher and subscriber (Python) — ROS 2 Documentation: Galactic documentation](https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html), [Recording a bag from a node (Python) — ROS 2 Documentation](https://docs.ros.org/en/galactic/Tutorials/Advanced/Recording-A-Bag-From-Your-Own-Node-Py.html) (Doing practical Python tutorials would be a great way to learn effectively.)

Hope you have now got a good understanding of how different tools of ros work.

# Understanding Kalman Filters:

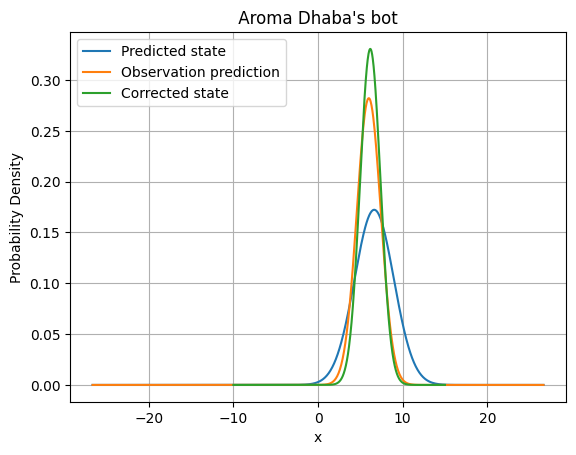
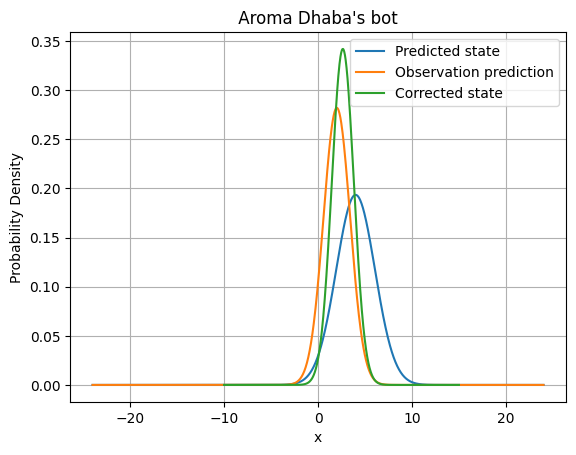
Now that you're familiar with the SLAM problem, it's worth noting that there exist various methods to tackle it. Among the earliest approaches is the utilization of Kalman filters and extended Kalman filters.

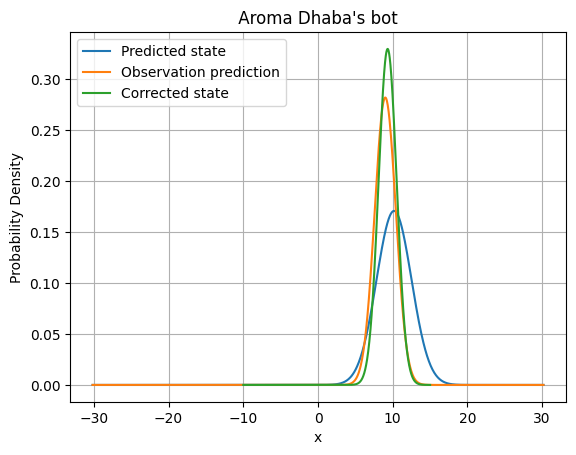
Kalman filter is based on [Recursive state estimation- Bayes filter](https://youtu.be/oUq0a8jHSQg?si=e6ObvFX45EjSlfZD) ( just a 5 min video 🙂)

[Kalman Filter & EKF (Cyrill Stachniss)](https://www.youtube.com/watch?v=E-6paM_Iwfc) (watch till 43:00)

Checkpoint 1:

1. You are tasked with coding a scenario where a food delivery robot moves from Aroma Dhaba (origin) to Hostel 3 in a 1-dimensional motion setting. The robot has sensors onboard that detect “Hostel” upon reaching any of the hostels (Hostel 1, 2, or 3, locations of which are known) and also senses the control command (speed). These sensors have some level of noise, and you need to use Kalman filters to estimate the robot's state accurately on the map. The goal is to ensure successful parcel delivery by localizing the robot's position accurately. We expect the final output to be like this:





Hint: Utilize loops to repeatedly send control commands and observations, following the Kalman algorithm iteratively.

# Moving to Extended Kalman Filters:

[Finish till end](https://youtu.be/E-6paM_Iwfc?si=H6chp8WCRk1p4yo3) 😀

Hope you enjoyed the algorithms (at least we didn’t 🙁)

Motion update and measurement update are two crucial terms in the field. Below is a brief explanation of each.

## Motion update:

The motion update step predicts the robot's new pose based on control inputs, such as velocity and angular rotation. It uses a motion model to estimate the expected change in the robot's pose. The motion model captures the kinematics or dynamics of the robot and takes into account factors like wheel odometry or inertial measurements.

During the motion update, the algorithm propagates the current estimate of the robot's pose by incorporating the control inputs. It predicts the new position and orientation of the robot based on the motion model. This prediction provides an initial estimate for the next step. It is expressed as [](https://www.codecogs.com/eqnedit.php?latex=%20P(x_t%20%7C%20x_%7Bt-1%7D%2C%20u_t%20)%20#0).

.[](https://www.codecogs.com/eqnedit.php?latex=%20P(x_t%20%7C%20x_%7Bt-1%7D%2C%20u_t%20)%20#0) - denotes the conditional probability distribution of the state variable xt at time t, **given** the previous state xt-1 and the control variable ut at time t.

## Measurement Update:

The measurement update step corrects the predicted pose estimate using sensor measurements. These measurements can be obtained from various sensors such as cameras, lidar, or range finders. The measurements capture the robot's perception of the environment, such as landmarks or features.

In the measurement update, the algorithm compares the predicted pose with the actual sensor measurements. It computes the likelihood or the probability of the measurements given the predicted pose. This likelihood is used to update the pose estimate using the Kalman filter equations. The update improves the accuracy of the estimated pose and allows for the creation of a map by associating the measurements with landmarks in the environment

It refers to the model describing the **expected observations given the current state** of the robot. It is expressed as [](https://www.codecogs.com/eqnedit.php?latex=%20P(z_t%7Cx_t)%20#0). Some examples include the Laser range bearing sensor model and Beam end-point model.

[](https://www.codecogs.com/eqnedit.php?latex=%20P(z_t%7Cx_t)%20#0) - denotes the conditional probability distribution of the observation variable zt at time t, given the state variable xt at the same time.

## 

Checkpoint 2:

Now that you know the basics, the next assignment is to fully code the Motion Update part of SLAM. Some pointers that may lead you to the solution:

* We trust you're familiar with the concept of rosbags. This task entails managing a rosbag to collect data from various sensors.
* Look in the [Formula Student site](https://fs-driverless.github.io/Formula-Student-Driverless-Simulator/v2.2.0/ros-bridge/#published-topics) for the topics published so that you can get the relevant topics. You can subscribe to get data. (also check topics published by ‘ros2 topic list’ and data by ‘ros2 topic echo TOPIC\_NAME’.
* You will require data from 2 topics simultaneously, so figure out how to do that
* Finally, visualize the car in rviz.

# What have we done till now?

Here is the portal for all of our team’s history: [TeamWIKI](https://iitb-racing.github.io/team-wiki/estimation/overview/)

# Data Association:

Data association is the crux of SLAM. As the robot navigates, its sensors constantly capture new information about the environment. Data association determines whether these observations correspond to existing landmarks in the map or if they represent entirely new features. This process is crucial for building an accurate map. Incorrect associations can lead the robot to believe it's revisited a location when it hasn't, causing errors in the map and the robot's estimated position. By correctly associating sensor data, SLAM can maintain a consistent understanding of the environment and the robot's place within it.

To learn the different types of data-association algorithms, kindly read these:

You don’t need to go into depth on their complexity, etc. Just try to get an understanding of their differences and how they work. Don’t get overwhelmed with all the data and confusing math in the papers, the goal is to get a gist of it.

<https://www.cds.caltech.edu/~murray/courses/me132-wi11/me132a_lec16.pdf> - what is data assoc? (Pg 1 - 43)

[Data Association in Stochastic Mapping using the Joint Compatibility Test](https://web.mit.edu/2.166/www/handouts/Neira_TRA_2001.pdf) - JCBB & ICNN

We are currently converting these algorithms to Python. Our team has, till date, converted ICNN into Python and is in the process of converting JCBB.

Checkpoint 3:

Think of a way of recognising and distinguishing the cones in our track. Write a Python code for your algorithm and run it on our full-lap rosbag.

Also read and attempt the first question of this document (Optional) : <https://www.cds.caltech.edu/~murray/courses/me132-wi11/lab02_v3.pdf>

The zip file required for the above document will be shared in the meeting.

We hope you guys enjoy this assignment.

Happy learning!!