ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

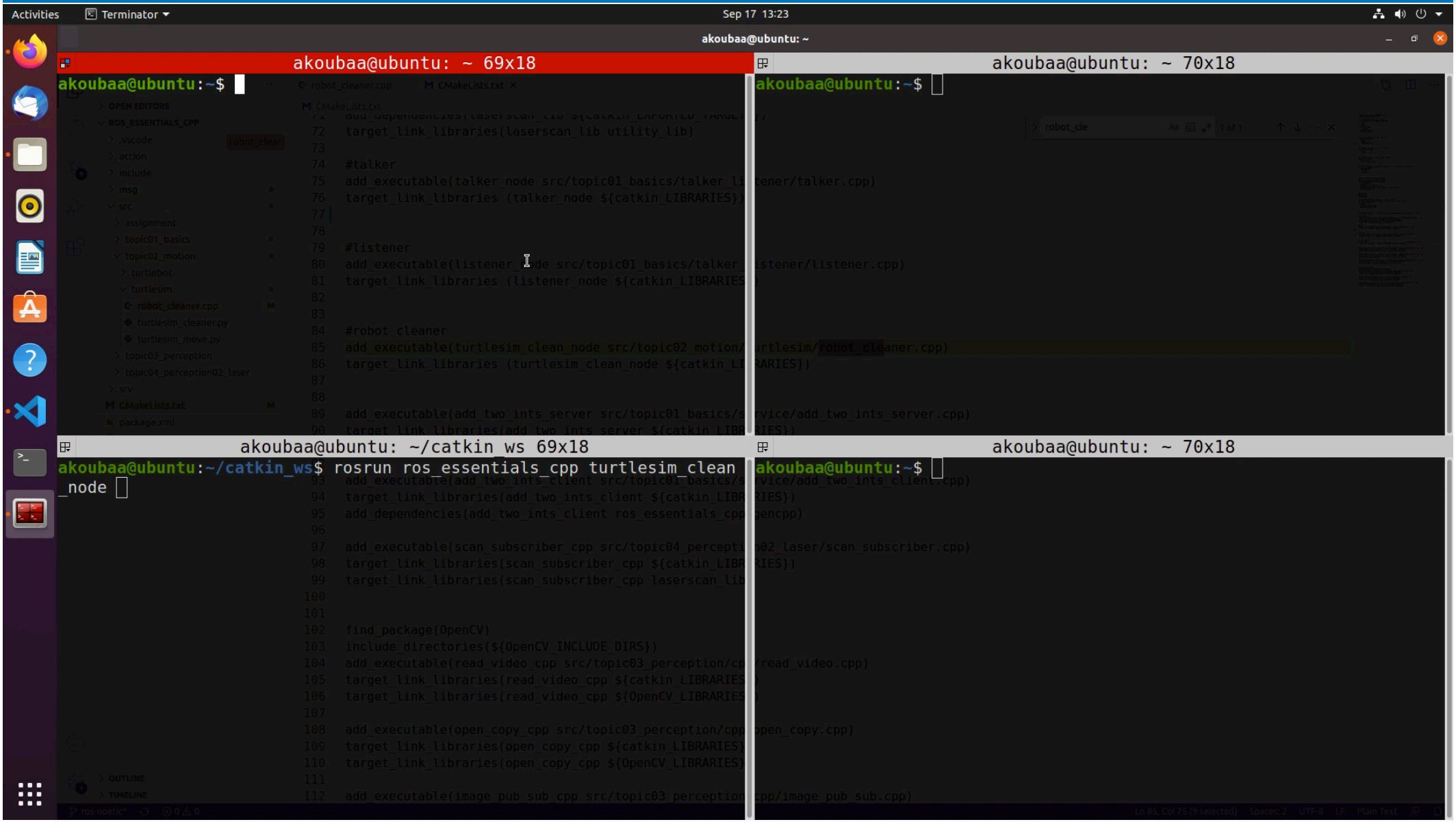
ANIS KOUBAA

ROS Motion

https://www.udemy.com/user/anis-koubaa/

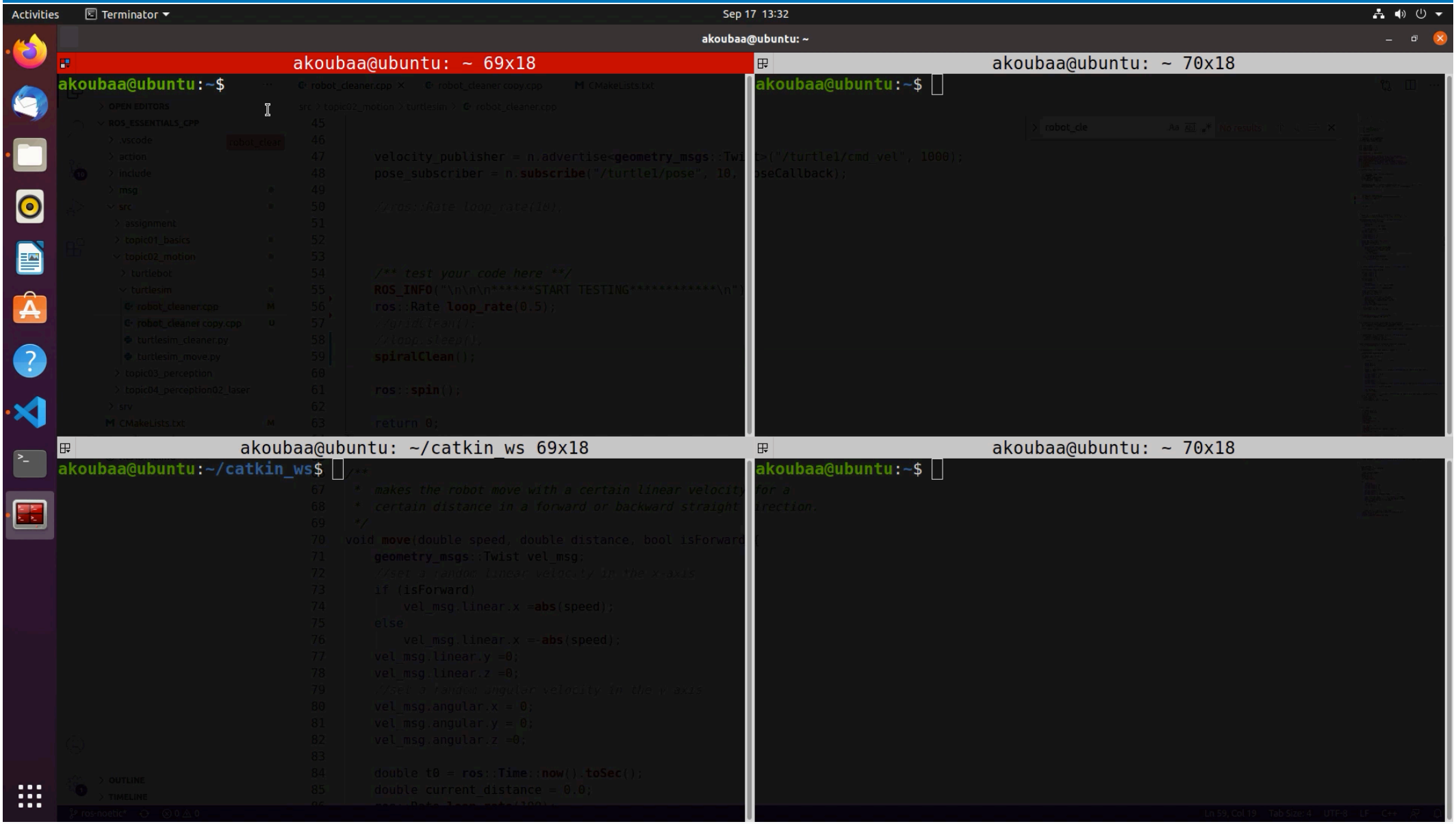
2020

Grid Trajectory



Anis Koubaa

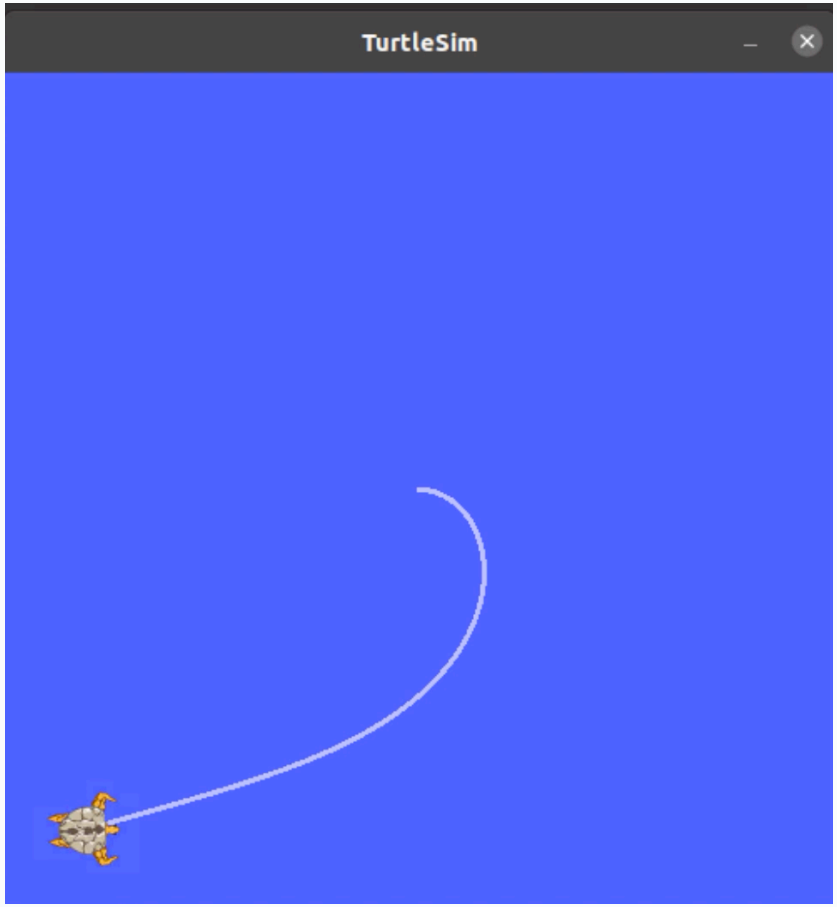
Spiral Trajectory



Anis Koubaa

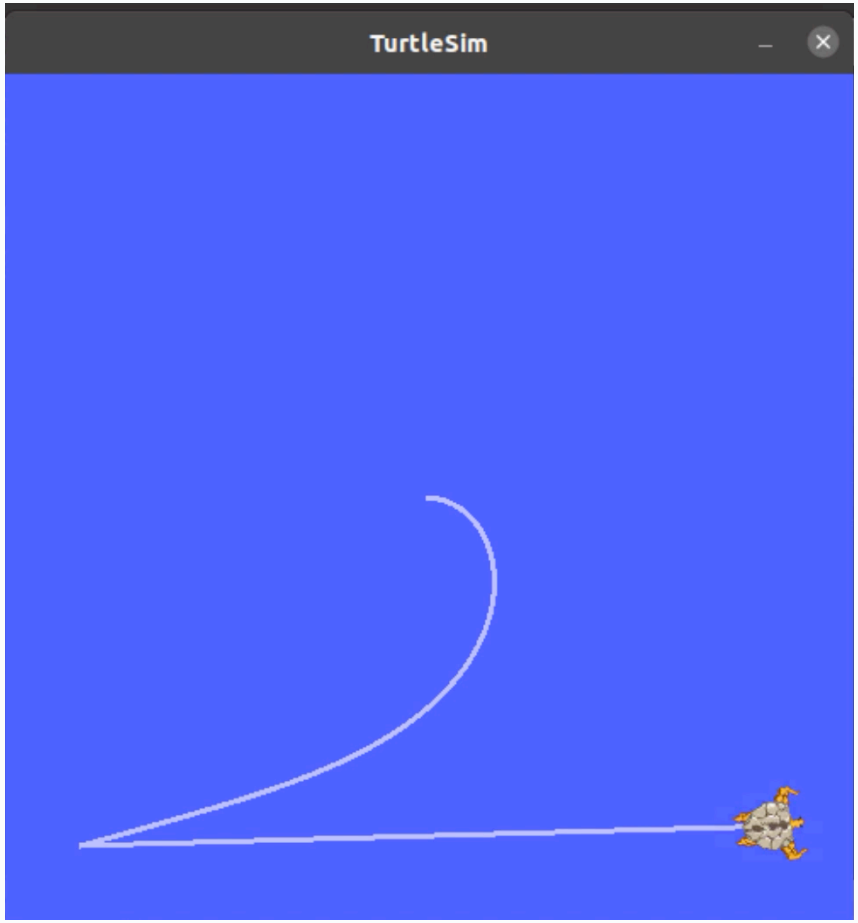
Basic Motion Types: Move in a Straight Line

```
Linear:
  x: speed
  y: 0
  z: 0

Angular:
  X: 0
  y: 0
  z: 0
```



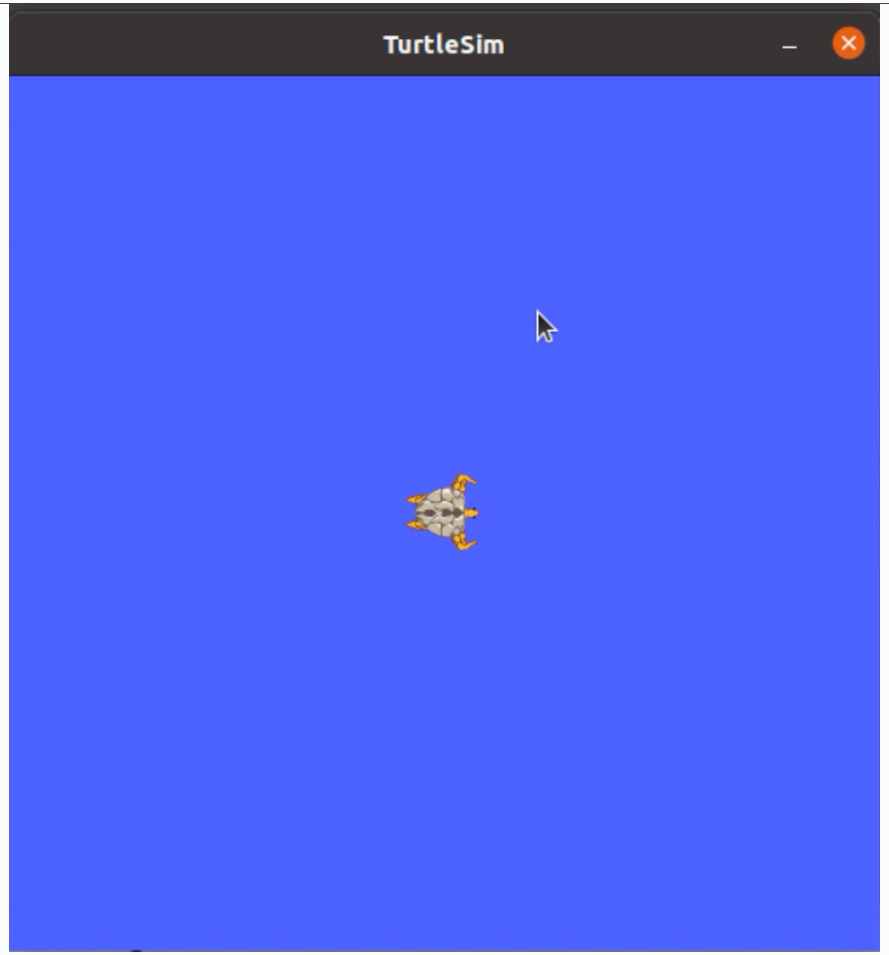
Basic Motion Types: Rotate in Place

```
Linear:
  x: 0
  y: 0
  z: 0

Angular:
  X: 0
  y: 0
  z: speed
```



Basic Motion Types: Go to Goal

```
Linear:
  x: f(distance)
  y: 0
  z: 0

Angular:
  X: 0
  y: 0
  z: f(angle)
```



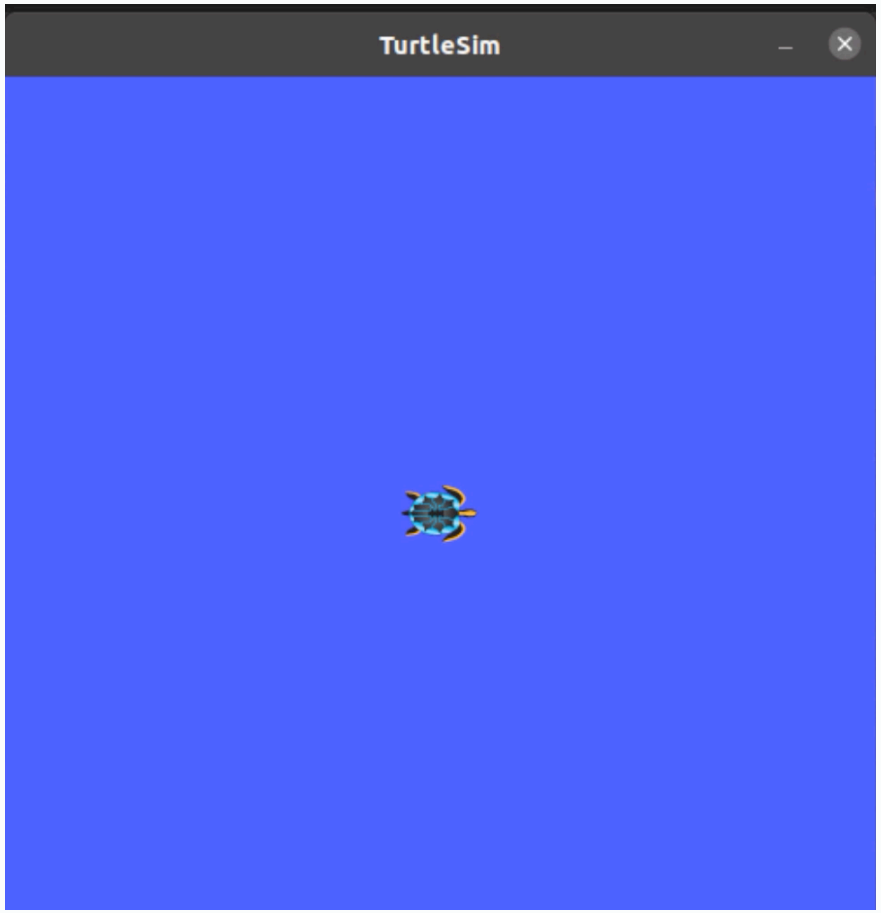
PID Controller

- Proportional
- Integral
- Derivative

Basic Motion Types: Spiral

Linear:
- x: *f(time)*
- y: *0*
- z: *0*

Angular:
- X: *0*
- y: *0*
- z: *constant*



ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

ANIS KOUBAA

Cleaning Application in Python and C++

https://www.udemy.com/user/anis-koubaa/

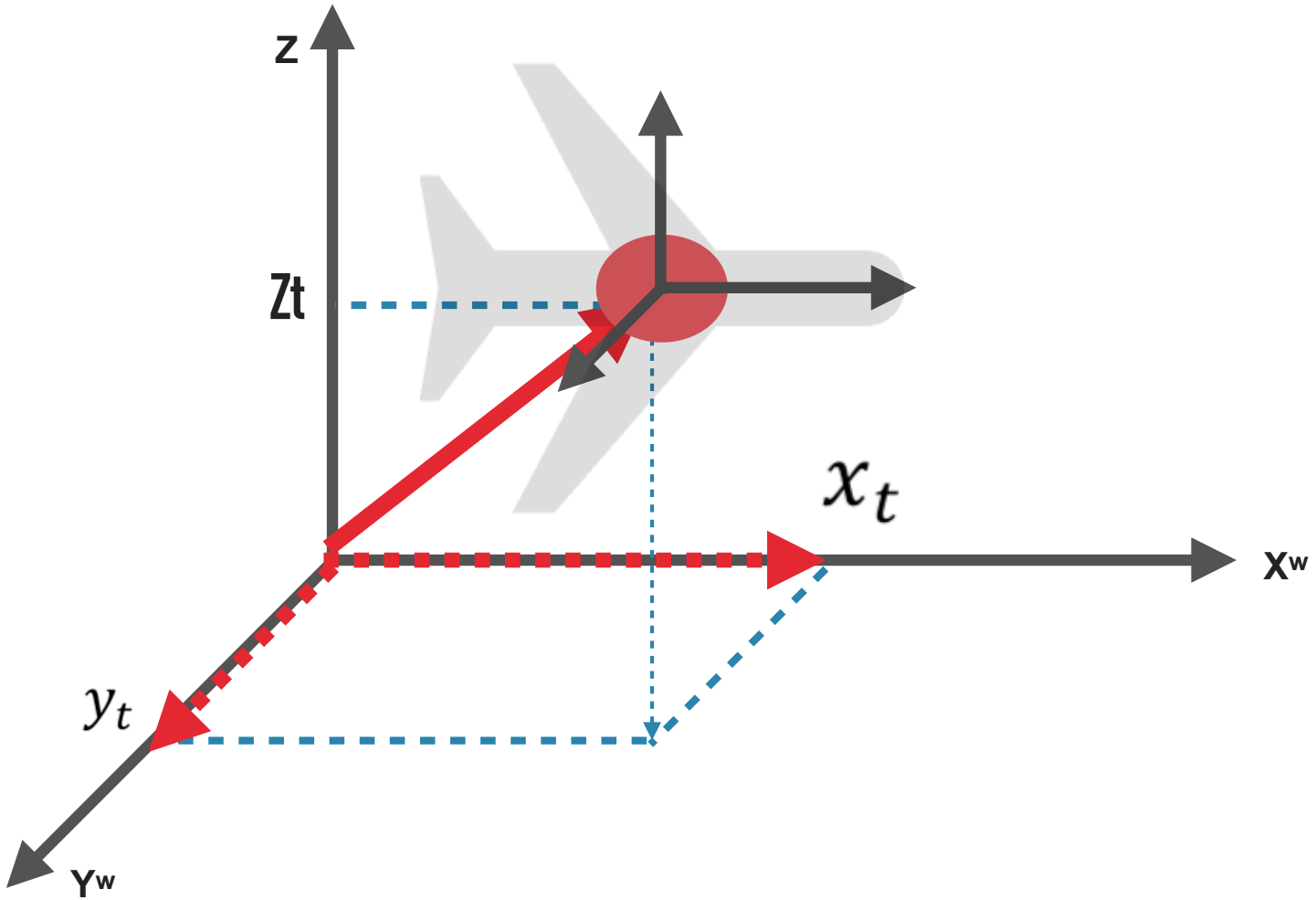
2020

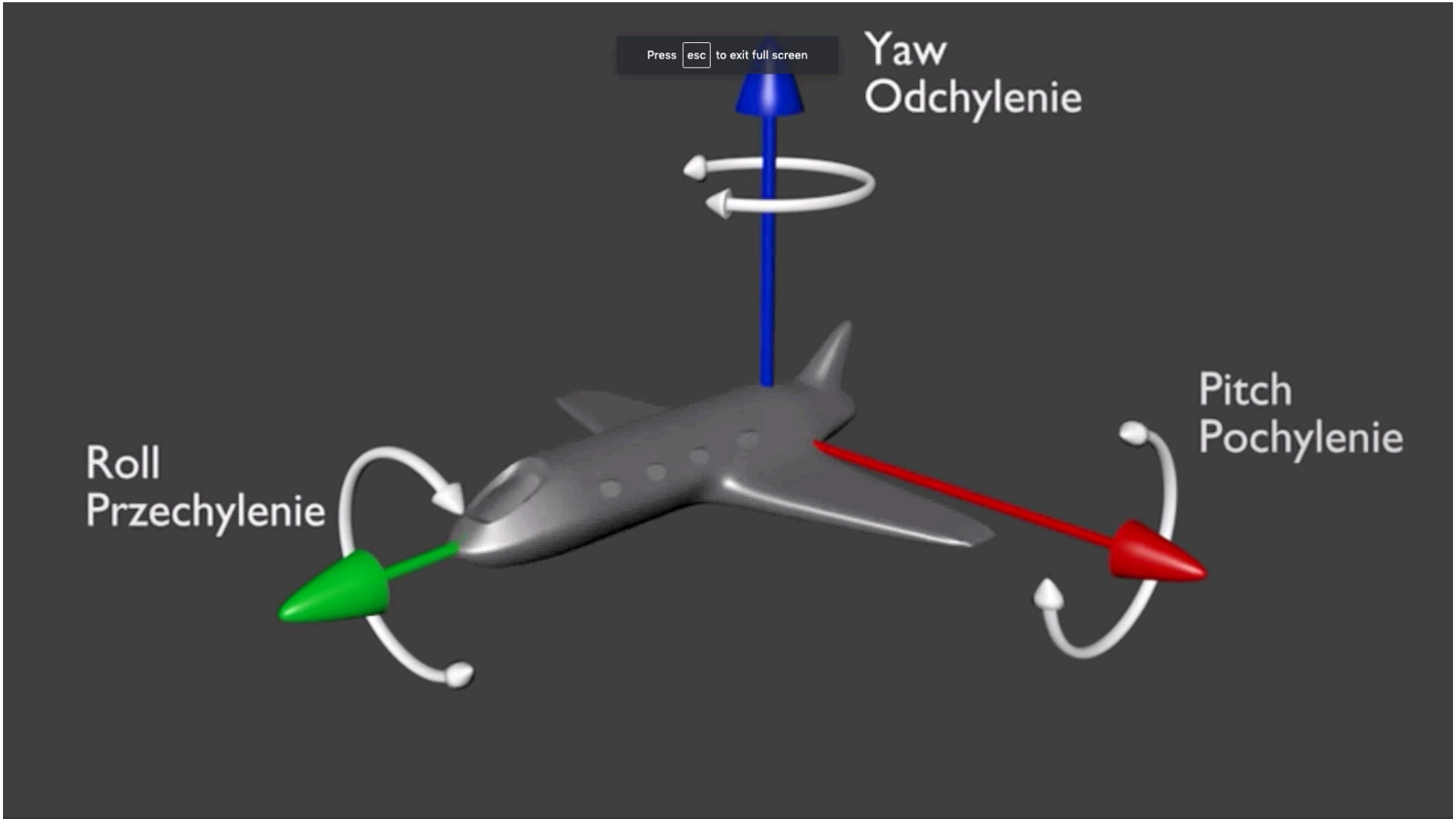
Step 1 : Understand topics and message to be used

```
akoubaa@ubuntu: ~/catkin_ws 69x26
akoubaa@ubuntu:~/catkin_ws$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
akoubaa@ubuntu:~/catkin_ws$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers: None

Subscribers:
 * /turtlesim (http://ubuntu:39313/)


akoubaa@ubuntu:~/catkin_ws$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z

akoubaa@ubuntu:~/catkin_ws$
```

Twist

Motion in Space



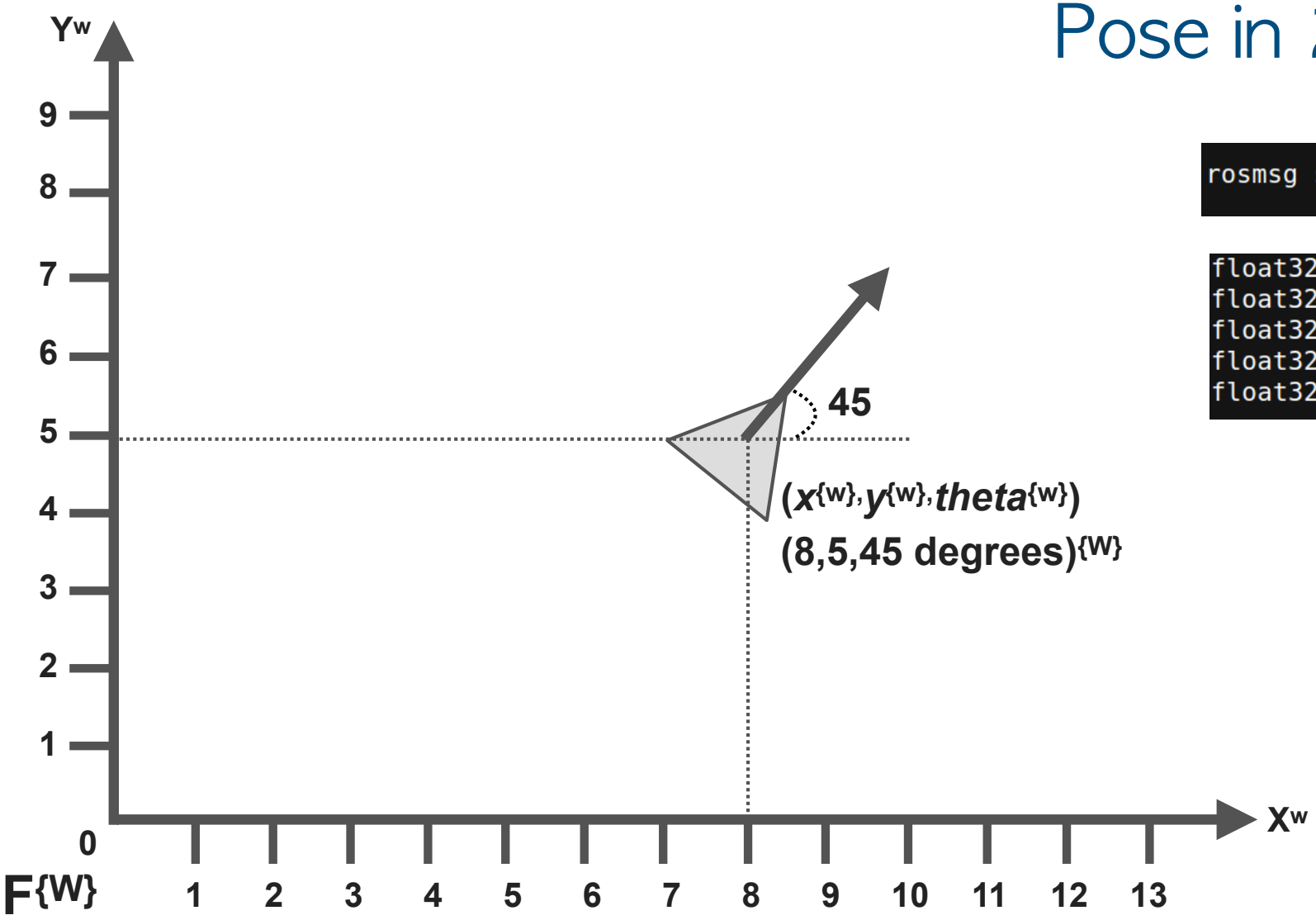

Press esc to exit full screen
Yaw
Odchylenie
Pitch
Pochylenie
Roll
Przechylenie

Step 1 : Understand topics and message to be used

```
akoubaa@ubuntu: ~/catkin_ws 69x26
akoubaa@ubuntu:~/catkin_ws$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
akoubaa@ubuntu:~/catkin_ws$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://ubuntu:39313/)

Subscribers: None


akoubaa@ubuntu:~/catkin_ws$ rosmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

akoubaa@ubuntu:~/catkin_ws$
```

Pose

Pose in 2D Space




```
rosmsg show turtlesim/Pose
```

```
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

```
Linear:
  x: speed
  y: 0
  z: 0

Angular:
  X: 0
  y: 0
  z: speed
```

Import Libraries of Messages

```python
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
```

```cpp
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"
```

Divide and Conquer Approach

- **Step 1.** Develop a function to move in a straight line for a certain distance, forward and backward
- **Step 2.** Develop a function to rotate in place for a certain angle, clockwise and counter-clockwise
- **Step 3.** Develop a function to go to a goal location
- **Step 4.** Develop a function to move in a spiral shape
- **Step 5.** Integrate all together to develop the cleaning application

ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

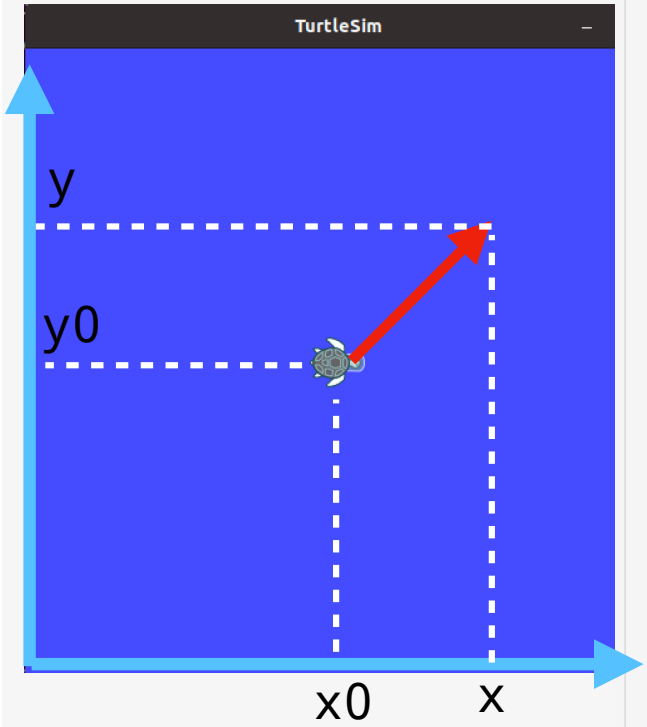
ANIS KOUBAA

Move Straight (C++/Python)

https://www.udemy.com/user/anis-koubaa/

2020

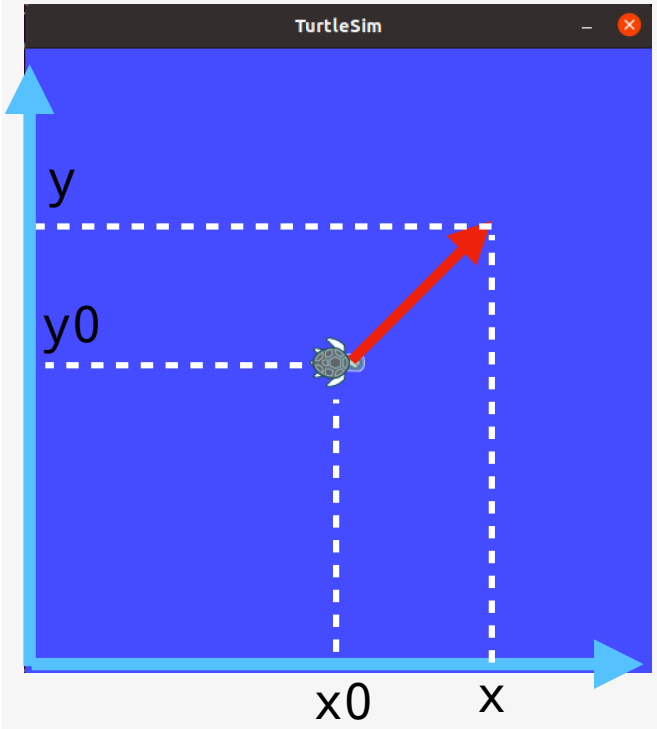
Move Straight




```python
def move(velocity_publisher, speed, distance, is_forward):
    #declare a Twist message to send velocity commands
    velocity_message = Twist()
    #get current location
    global x, y
    x0=x #save the initial location x-coordinate
    y0=y #save the initial location y-coordinate

    if (is_forward):
        velocity_message.linear.x =abs(speed)
    else:
        velocity_message.linear.x =-abs(speed)

    distance_moved = 0.0
    loop_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times a second)

    while True :
            rospy.loginfo("Turtlesim moves forwards")
            velocity_publisher.publish(velocity_message)
            loop_rate.sleep()
            distance_moved = abs(      math.sqrt(((x-x0) ** 2) + ((y-y0) ** 2)))
            print  distance_moved
            if  not (distance_moved<distance):
                rospy.loginfo("reached")
                break

    #finally, stop the robot when the distance is moved
    velocity_message.linear.x =0
    velocity_publisher.publish(velocity_message)
```

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

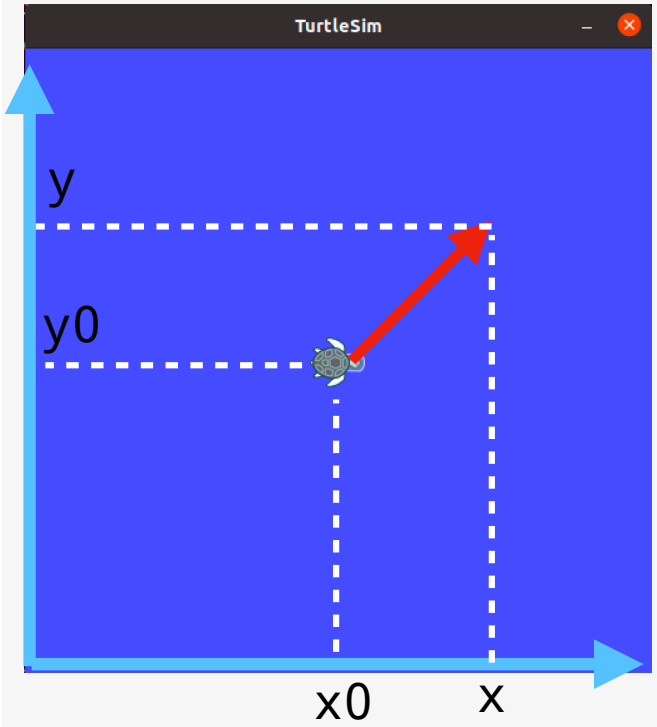
Move Straight



Create a Pose Callback

```python
def poseCallback(pose_message):
    global x
    global y, yaw
    x= pose_message.x
    y= pose_message.y
    yaw = pose_message.theta

if __name__ == '__main__':
    try:

        rospy.init_node('turtlesim_motion_pose', anonymous=True)

        #declare velocity publisher
        cmd_vel_topic='/turtle1/cmd_vel'
        velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)

        position_topic = "/turtle1/pose"
        pose_subscriber = rospy.Subscriber(position_topic, Pose, poseCallback)
        time.sleep(2)
```

Move Straight

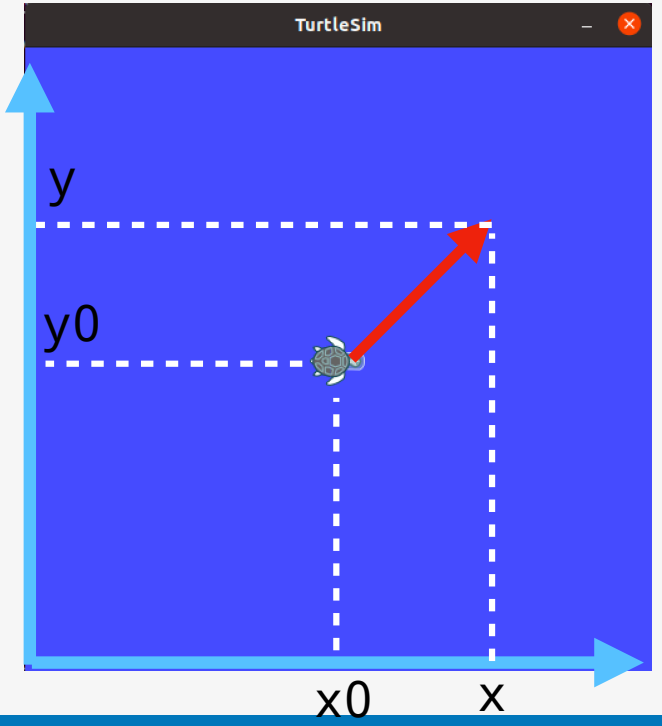



```
void move(double speed, double distance, bool isForward){
    geometry_msgs::Twist vel_msg;
    //set a random linear velocity in the x-axis
    if (isForward)
        vel_msg.linear.x =abs(speed);
    else
        vel_msg.linear.x =-abs(speed);

    double t0 = ros::Time::now().toSec();
    double current_distance = 0.0;
    ros::Rate loop_rate(100);
    do{
        velocity_publisher.publish(vel_msg);
        double t1 = ros::Time::now().toSec();
        current_distance = speed * (t1-t0);
        ros::spinOnce();
        loop_rate.sleep();
        //cout<<(t1-t0)<<", "<<current_distance <<", "<<distance<<endl;
    }while(current_distance<distance);
    vel_msg.linear.x =0;
    velocity_publisher.publish(vel_msg);

}
```

Move Straight

```
void poseCallback(const turtlesim::Pose::ConstPtr & pose_message){
    turtlesim_pose.x=pose_message->x;
    turtlesim_pose.y=pose_message->y;
    turtlesim_pose.theta=pose_message->theta;
}
```

```
int main(int argc, char **argv)
{
    // Initiate new ROS node named "talker"
    ros::init(argc, argv, "turtlesim_cleaner");
    ros::NodeHandle n;
    double speed, angular_speed;
    double distance, angle;
    bool isForward, clockwise;


    velocity_publisher = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1000);
    pose_subscriber = n.subscribe("/turtle1/pose", 10, poseCallback);

    /** test your code here **/
    ROS_INFO("\n\n\n******START TESTING**********\n");
    ros::Rate loop_rate(0.5);
    gridClean();
    //loop.sleep();
    //spiralClean();

    ros::spin();

    return 0;
}
```



ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

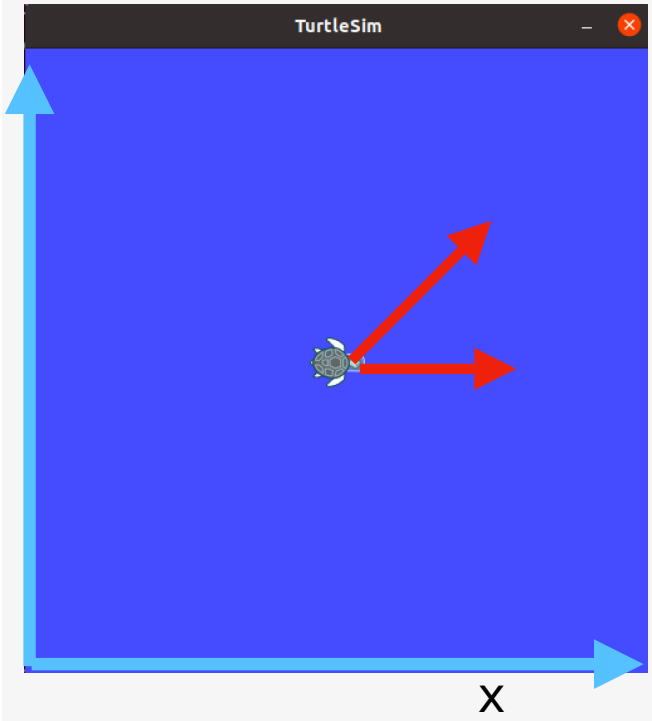
ANIS KOUBAA

Rotate (C++/Python)

https://www.udemy.com/user/anis-koubaa/

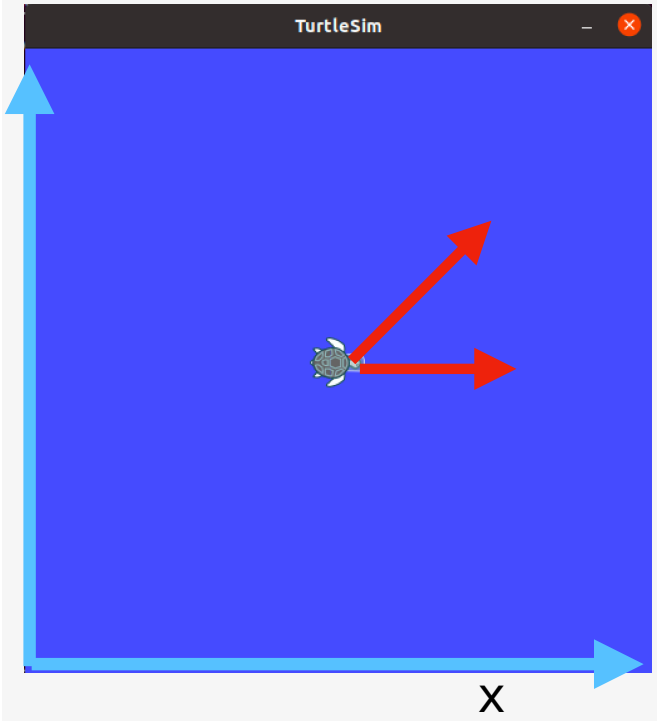
2020

Rotate




```python
def rotate (velocity_publisher, angular_speed_degree, relative_angle_degree, clockwise):

    velocity_message = Twist()

    angular_speed=math.radians(abs(angular_speed_degree))

    if (clockwise):
        velocity_message.angular.z =-abs(angular_speed)
    else:
        velocity_message.angular.z =abs(angular_speed)

    loop_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times a second)
    t0 = rospy.Time.now().to_sec()

    while True :
        rospy.loginfo("Turtlesim rotates")
        velocity_publisher.publish(velocity_message)

        t1 = rospy.Time.now().to_sec()
        current_angle_degree = (t1-t0)*angular_speed_degree
        loop_rate.sleep()

        if  (current_angle_degree>relative_angle_degree):
            rospy.loginfo("reached")
            break

    #finally, stop the robot when the distance is moved
    velocity_message.angular.z =0
    velocity_publisher.publish(velocity_message)
```

Rotate




```cpp
void rotate (double angular_speed, double relative_angle, bool clockwise){

    geometry_msgs::Twist vel_msg;

    if (clockwise)
        vel_msg.angular.z =-abs(angular_speed);
    else
        vel_msg.angular.z =abs(angular_speed);

    double current_angle = 0.0;
    double t0 = ros::Time::now().toSec();
    ros::Rate loop_rate(10);
    do{
        velocity_publisher.publish(vel_msg);
        double t1 = ros::Time::now().toSec();
        current_angle = angular_speed * (t1-t0);
        ros::spinOnce();
        loop_rate.sleep();
    }while(current_angle<relative_angle);

    vel_msg.angular.z =0;
    velocity_publisher.publish(vel_msg);

}
```

ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

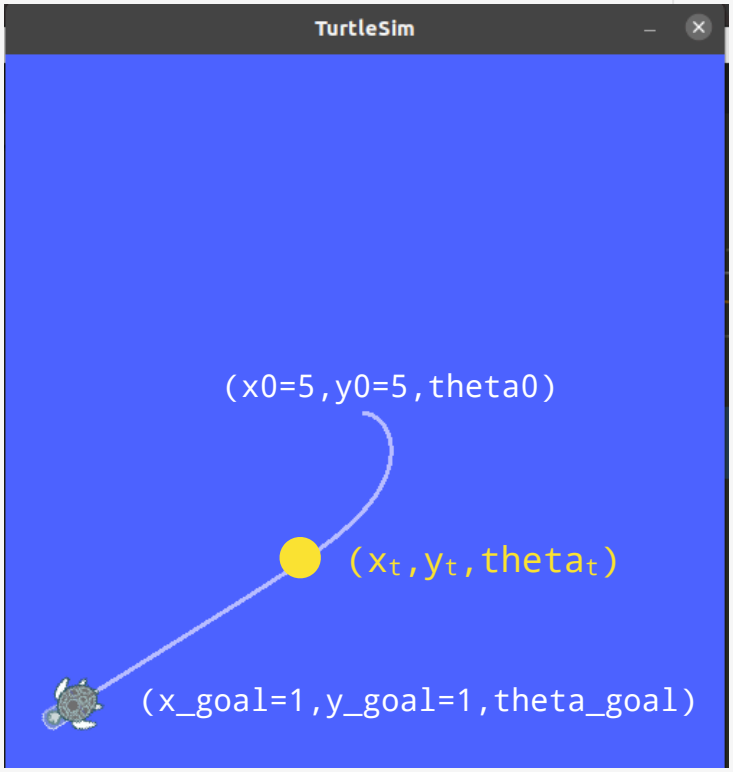
ANIS KOUBAA

Go to Goal (C++/Python)

Feedback Control

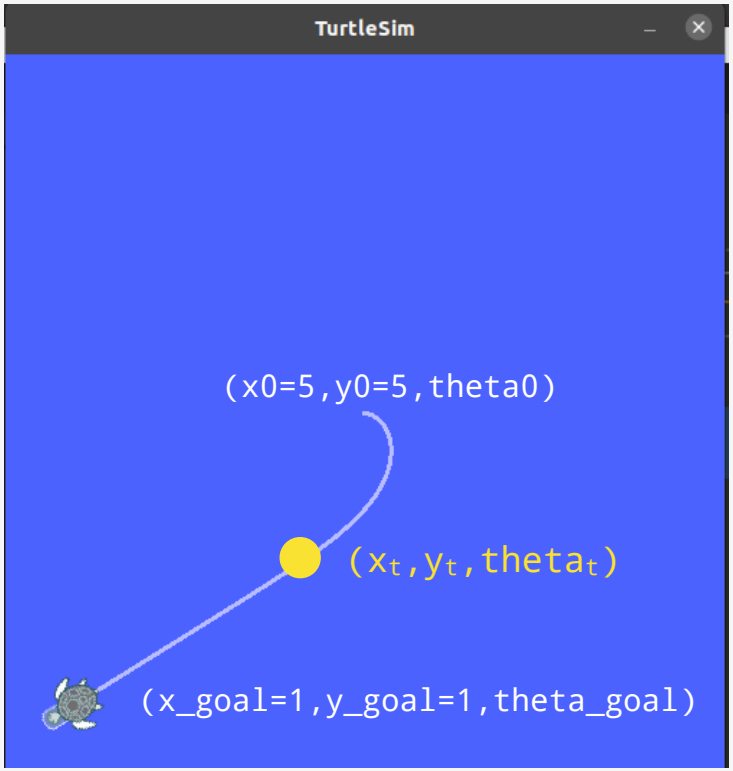
https://www.udemy.com/user/anis-koubaa/
2020

Go-To-Goal




```python
def go_to_goal(velocity_publisher, x_goal, y_goal):
    global x
    global y, yaw

    velocity_message = Twist()

    while (True):
        K_linear = 0.5
        distance = abs(math.sqrt(((x_goal-x) ** 2) + ((y_goal-y) ** 2)))

        linear_speed = distance * K_linear

        K_angular = 4.0
        desired_angle_goal = math.atan2(y_goal-y, x_goal-x)
        angular_speed = (desired_angle_goal-yaw)*K_angular

        velocity_message.linear.x = linear_speed
        velocity_message.angular.z = angular_speed

        velocity_publisher.publish(velocity_message)
        print ('x=', x, ', y=',y, ', distance to goal: ', distance)

        if (distance <0.01):
            break
```

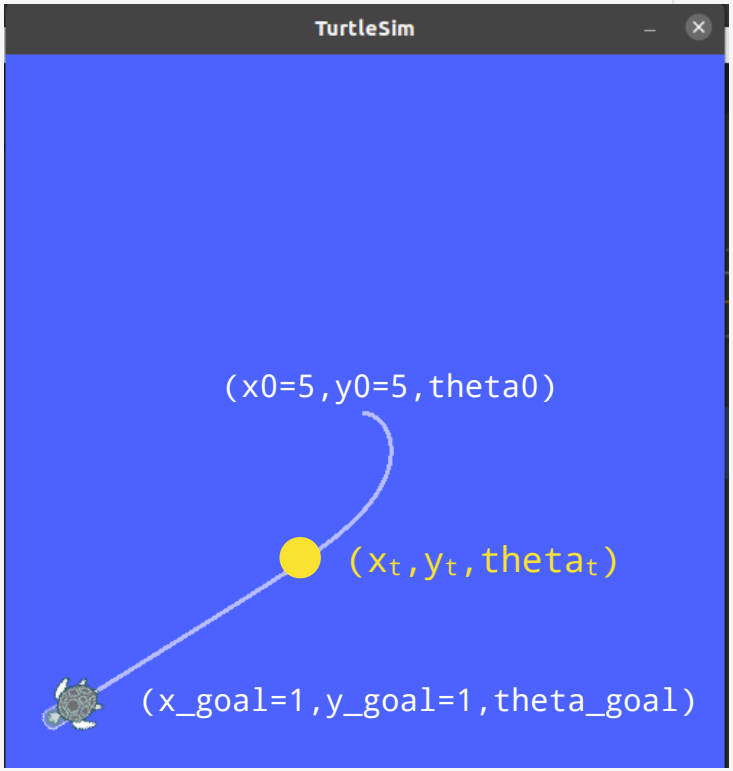
Go-To-Goal



Create a Pose Callback

```
def poseCallback(pose_message):
    global x
    global y, yaw
    x= pose_message.x
    y= pose_message.y
    yaw = pose_message.theta

if __name__ == '__main__':
    try:

        rospy.init_node('turtlesim_motion_pose', anonymous=True)

        #declare velocity publisher
        cmd_vel_topic='/turtle1/cmd_vel'
        velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)

        position_topic = "/turtle1/pose"
        pose_subscriber = rospy.Subscriber(position_topic, Pose, poseCallback)
        time.sleep(2)
```

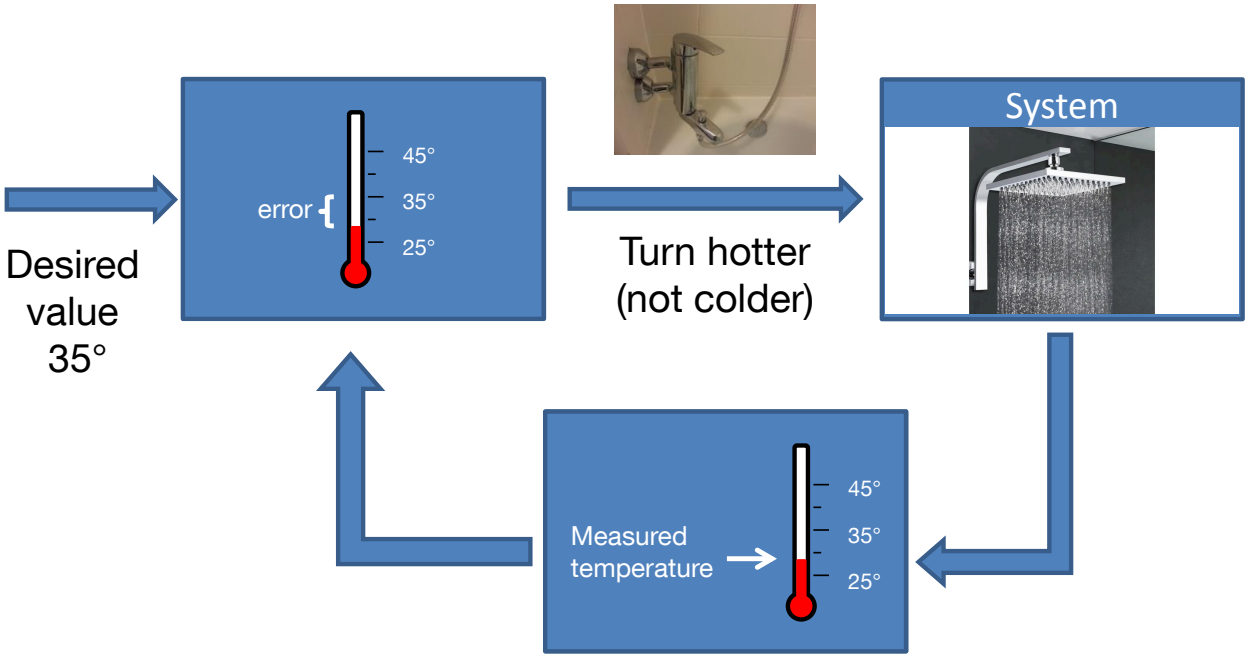
Go-To-Goal



PID Controller

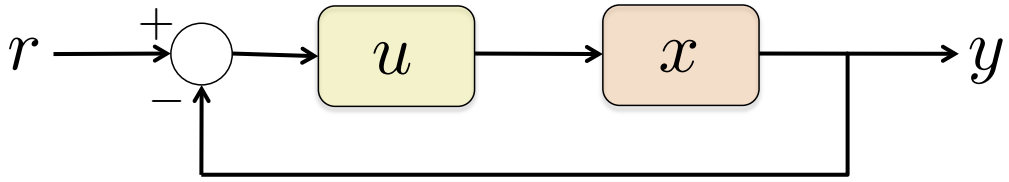
```
def go_to_goal(velocity_publisher, x_goal, y_goal):
    global x
    global y, yaw

    velocity_message = Twist()

    while (True):
        K_linear = 0.5
        distance = abs(math.sqrt(((x_goal-x) ** 2) + ((y_goal-y) ** 2)))

        linear_speed = distance * K_linear

        K_angular = 4.0
        desired_angle_goal = math.atan2(y_goal-y, x_goal-x)
        angular_speed = (desired_angle_goal-yaw)*K_angular

        velocity_message.linear.x = linear_speed
        velocity_message.angular.z = angular_speed

        velocity_publisher.publish(velocity_message)
        print ('x=', x, ', y=',y, ', distance to goal: ', distance)

        if (distance <0.01):
            break
```

Feedback Control

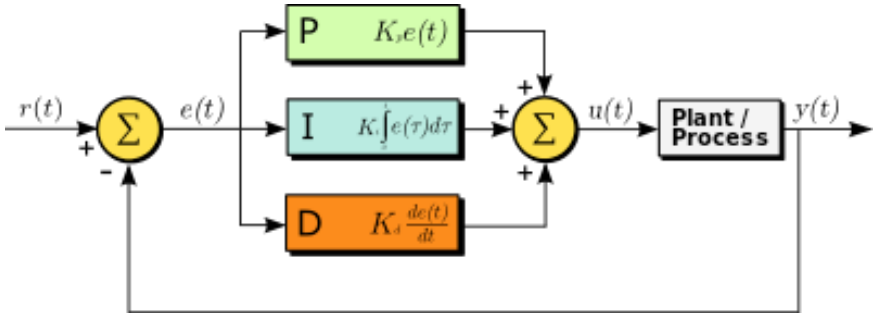


Feedback Control

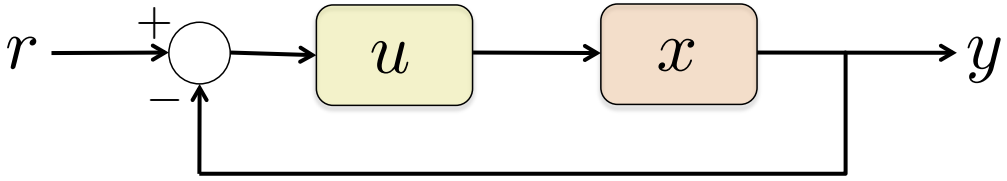
- **State:** Representation of what the system is currently doing
- **Dynamics:** Description of how the state changes
- **Reference:** What we want the system to do
- **Output:** Measurement of (some aspects of the) system
- **Input:** Control signal produced with respect to the reference
- **Feedback:** Mapping from outputs to inputs



Feedback Control: Pose Control



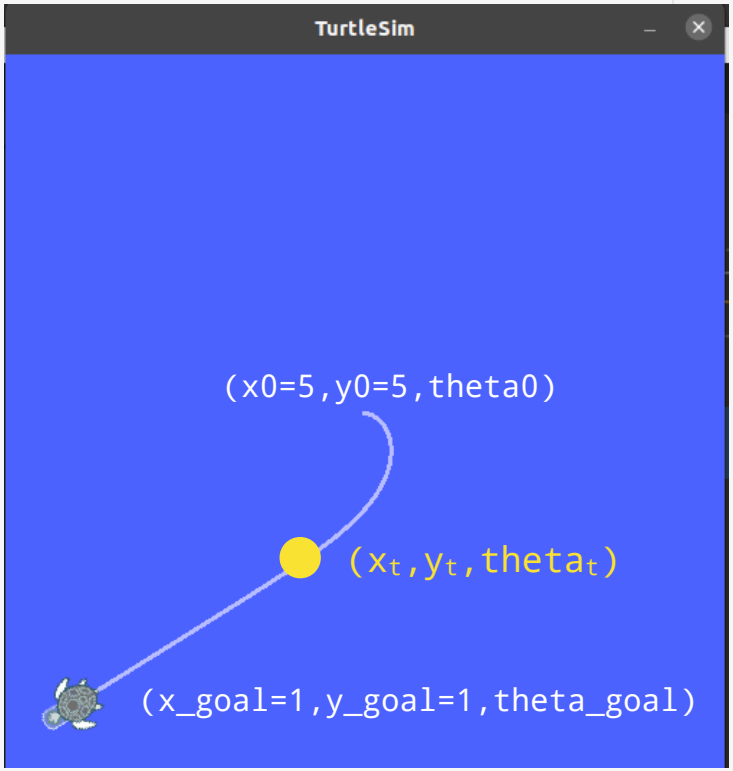
- **State:** the pose of the system **(x,y,theta)**
- **Dynamics:** `speed = x * time`
- **Reference:** move to goal location **(x_{goal},y_{goal},$theta_{goal}$)**
- **Output:**
  - `Distance Error = sqrt((`x_{goal}`-x)`^{2}`+(`y_{goal}`-y)`^{2}`)` $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
  - `Angle Error =` $theta_{goal}$ `- theta`
- **Input:** Linear Velocity **(proportional to distance error)** and Angular Velocity **(proportional to angle error)**
- **Feedback:** Adapt linear velocity and angular velocity based on the error



Go-To-Goal

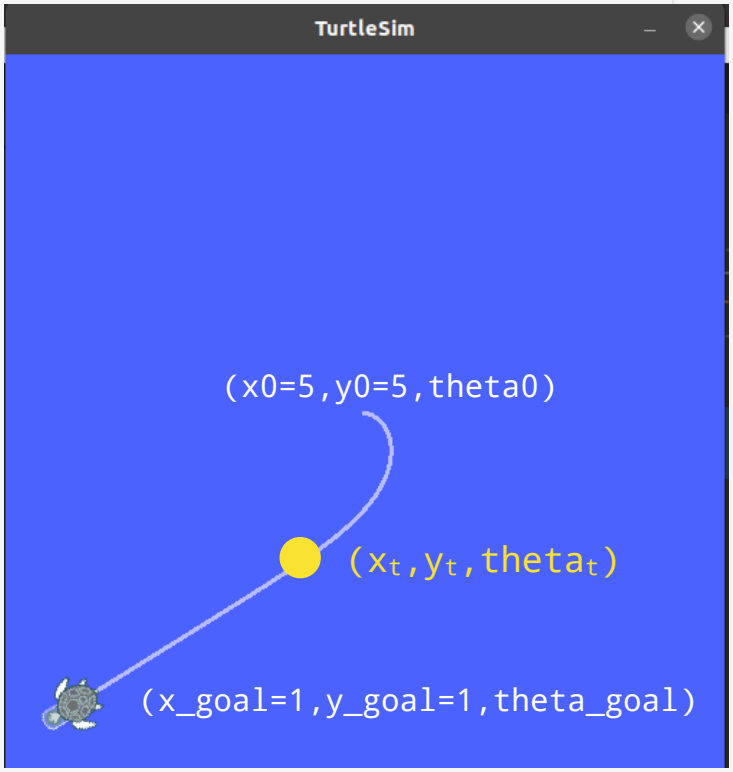
P-Controller

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$




```python
def go_to_goal(velocity_publisher, x_goal, y_goal):
    global x
    global y, yaw

    velocity_message = Twist()

    while (True):
        K_linear = 0.5
        distance = abs(math.sqrt(((x_goal-x) ** 2) + ((y_goal-y) ** 2)))

        linear_speed = distance * K_linear

        K_angular = 4.0
        desired_angle_goal = math.atan2(y_goal-y, x_goal-x)
        angular_speed = (desired_angle_goal-yaw)*K_angular

        velocity_message.linear.x = linear_speed
        velocity_message.angular.z = angular_speed

        velocity_publisher.publish(velocity_message)
        print ('x=', x, ', y=',y, ', distance to goal: ', distance)

        if (distance <0.01):
            break
```

Go-To-Goal



P-Controller

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
def go_to_goal(velocity_publisher, x_goal, y_goal):
    global x
    global y, yaw

    velocity_message = Twist()

    while (True):
        K_linear = 0.5
        distance = abs(math.sqrt(((x_goal-x) ** 2) + ((y_goal-y) ** 2)))

        linear_speed = distance * K_linear

        K_angular = 4.0
        desired_angle_goal = math.atan2(y_goal-y, x_goal-x)
        angular_speed = (desired_angle_goal-yaw)*K_angular

        velocity_message.linear.x = linear_speed
        velocity_message.angular.z = angular_speed

        velocity_publisher.publish(velocity_message)
        print ('x=', x, ', y=',y, ', distance to goal: ', distance)

        if (distance <0.01):
            break
```

Go-To-Goal

P-Controller

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
void moveGoal(turtlesim::Pose  goal_pose, double distance_tolerance){

    geometry_msgs::Twist vel_msg;

    ros::Rate loop_rate(100);
    double E = 0.0;
    do{
        /****** Proportional Controller ******/
        //linear velocity in the x-axis
        double Kp=1.0;
        double Ki=0.02;
        //double v0 = 2.0;
        //double alpha = 0.5;
        double e = getDistance(turtlesim_pose.x, turtlesim_pose.y, goal_pose.x, goal_pose.y);
        double E = E+e;
        //Kp = v0 * (exp(-alpha)*error*error)/(error*error);
        vel_msg.linear.x = (Kp*e);
        vel_msg.angular.z =4*(atan2(goal_pose.y-turtlesim_pose.y, goal_pose.x-turtlesim_pose.x)-turtlesim_pose.theta)

        velocity_publisher.publish(vel_msg);

        ros::spinOnce();
        loop_rate.sleep();

    }while(getDistance(turtlesim_pose.x, turtlesim_pose.y, goal_pose.x, goal_pose.y)>distance_tolerance);
    cout<<"end move goal"<<endl;
    vel_msg.linear.x =0;
    vel_msg.angular.z = 0;
    velocity_publisher.publish(vel_msg);
}
```

Go-To-Goal

```
void poseCallback(const turtlesim::Pose::ConstPtr & pose_message){
    turtlesim_pose.x=pose_message->x;
    turtlesim_pose.y=pose_message->y;
    turtlesim_pose.theta=pose_message->theta;
}
```

```
int main(int argc, char **argv)
{
    // Initiate new ROS node named "talker"
    ros::init(argc, argv, "turtlesim_cleaner");
    ros::NodeHandle n;
    double speed, angular_speed;
    double distance, angle;
    bool isForward, clockwise;


    velocity_publisher = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1000);
    pose_subscriber = n.subscribe("/turtle1/pose", 10, poseCallback);

    /** test your code here **/
    ROS_INFO("\n\n\n******START TESTING**********\n");
    ros::Rate loop_rate(0.5);
    gridClean();
    //loop.sleep();
    //spiralClean();

    ros::spin();

    return 0;
}
```

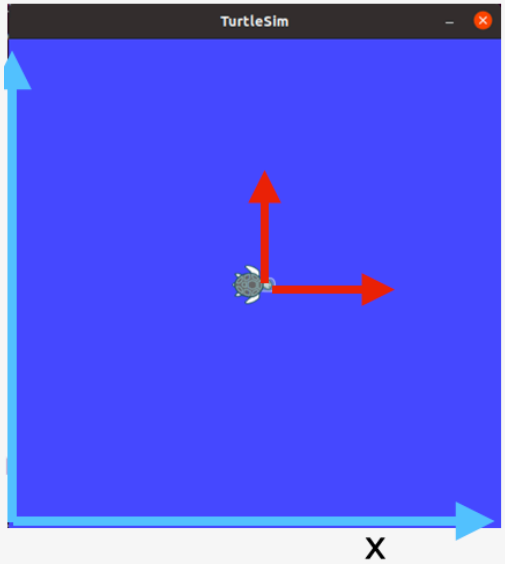
ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

ANIS KOUBAA

Set Desired Location

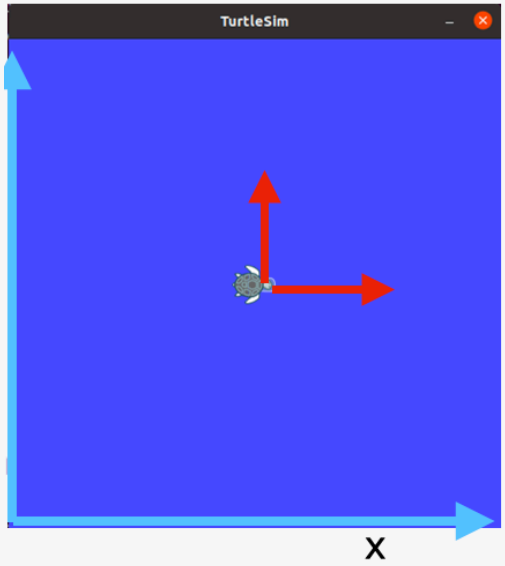
https://www.udemy.com/user/anis-koubaa/
2020

Set Desired Location




```python
def setDesiredOrientation(publisher, speed_in_degree, desired_angle_degree):
    relative_angle_radians = math.radians(desired_angle_degree) - yaw
    clockwise=0
    if relative_angle_radians < 0:
        clockwise = 1
    else:
        clockwise = 0
    print ("relative_angle_radians: ",math.degrees(relative_angle_radians))
    print ("desired_angle_degree: ",desired_angle_degree)
    rotate(publisher, speed_in_degree,math.degrees(abs(relative_angle_radians)), clockwise)
```

Set Desired Location




```
void setDesiredOrientation (double desired_angle_radians){
    double relative_angle_radians = desired_angle_radians - turtlesim_pose.theta;
    bool clockwise = ((relative_angle_radians<0)?true:false);
    rotate (degrees2radians(10), abs(relative_angle_radians), clockwise);
}
```

ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

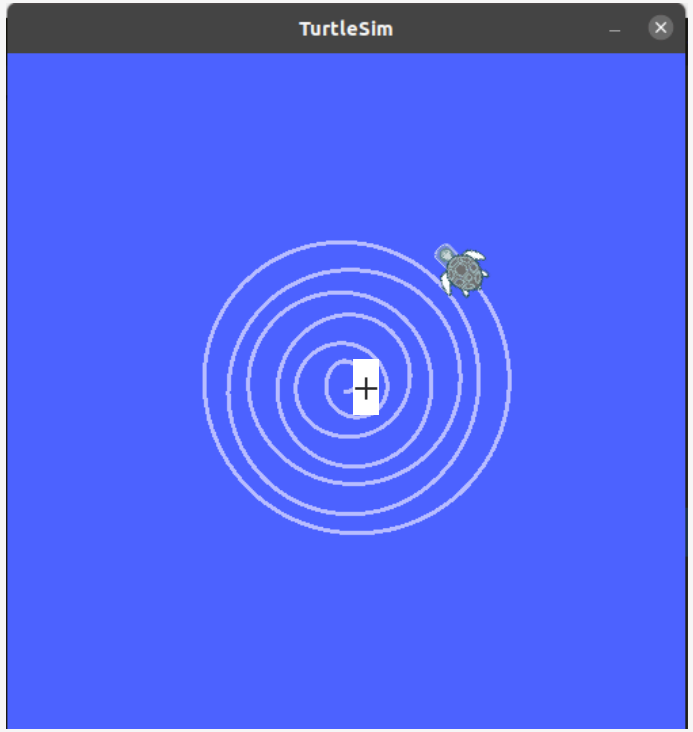
ANIS KOUBAA

Spiral Motion

https://www.udemy.com/user/anis-koubaa/
2020

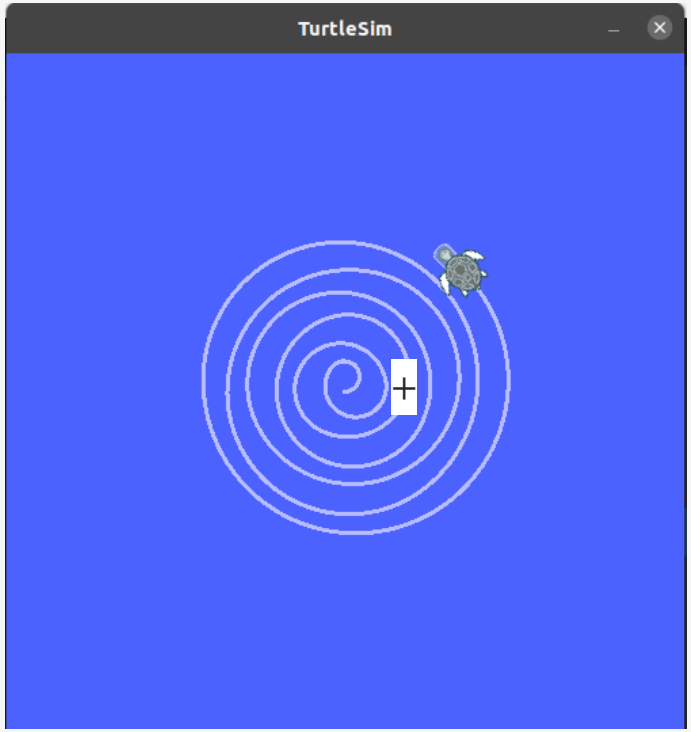
Spiral Motion

$$r = a + b\theta.$$




```python
def spiral(velocity_publisher, wk, rk):
    vel_msg = Twist()
    loop_rate = rospy.Rate(1)

    while((x<10.5) and (y<10.5)):
        rk=rk+1
        vel_msg.linear.x =rk
        vel_msg.linear.y =0
        vel_msg.linear.z =0
        vel_msg.angular.x = 0
        vel_msg.angular.y = 0
        vel_msg.angular.z =wk
        velocity_publisher.publish(vel_msg)
        loop_rate.sleep()

    vel_msg.linear.x = 0
    vel_msg.angular.z = 0
    velocity_publisher.publish(vel_msg)
```

Spiral Motion




```cpp
void spiralClean(double rk, double wk){
    geometry_msgs::Twist vel_msg;

    ros::Rate loop(1);

    do{
        rk=rk+1.0;
        vel_msg.linear.x =rk;
        vel_msg.angular.z =wk;

        cout<<"vel_msg.linear.x = "<<vel_msg.linear.x<<endl;
        cout<<"vel_msg.angular.z = "<<vel_msg.angular.z<<endl;
        velocity_publisher.publish(vel_msg);
        ros::spinOnce();

        loop.sleep();

    }while((turtlesim_pose.x<10.5)&&(turtlesim_pose.y<10.5));
    vel_msg.linear.x =0;
    velocity_publisher.publish(vel_msg);
}
```

ROS FOR BEGINNERS
BASICS, MOTION AND OPENCV

ANIS KOUBAA

Grid Cleaning

https://www.udemy.com/user/anis-koubaa/
2020

Grid Cleaning

```python
def gridClean(publisher):

    desired_pose = Pose()
    desired_pose.x = 1
    desired_pose.y = 1
    desired_pose.theta = 0

    go_to_goal(publisher, 1,1)

    setDesiredOrientation(publisher, 30, math.radians(desired_pose.theta))

    for i in range(5):
        move(publisher, 2.0, 1.0, True)
        rotate(publisher, 20, 90, False)
        move(publisher, 2.0, 9.0, True)
        rotate(publisher, 20, 90, True)
        move(publisher, 2.0, 1.0, True)
        rotate(publisher, 20, 90, True)
        move(publisher, 2.0, 9.0, True)
        rotate(publisher, 20, 90, False)
    pass
```

Grid Cleaning

```
void gridClean(){

    ros::Rate loop(0.5);
    turtlesim::Pose pose;
    pose.x=1;
    pose.y=1;
    pose.theta=0;
    moveGoal(pose, 0.01);
    loop.sleep();
    setDesiredOrientation(0);
    loop.sleep();

    move(2.0, 9.0, true);
    loop.sleep();
    rotate(degrees2radians(10), degrees2radians(90), false);
    loop.sleep();
    move(2.0, 9.0, true);


    rotate(degrees2radians(10), degrees2radians(90), false);
    loop.sleep();
    move(2.0, 1.0, true);
    rotate(degrees2radians(10), degrees2radians(90), false);
    loop.sleep();
    move(2.0, 9.0, true);
```