Telerobotics: UWSim
Assignment : Waypoints and Vision Control

Emanuele Sansebastiano

January 2017

1 Introduction

The simulator UWSim [2] is developed by the university Jaume I to simulate underwater robotic operations, like object recovery or underwater manipulations. This activities are performed by the robot called "Girona 500" [3] for the European project "Triton". Attached to this report there are to programs controlling the robot along an underwater pipe. The maintenance of underwater structures, like pipes, is really expansive, even if the broken part location is known. Sending humans underwater costs more than sending autonomous robots. The first of the two programs, called *"PF_waypoints_sansebastiano.py"*, bases its control law on knowing the waypoints the robot has to span. The second one, called *"PF_vision_sansebastiano.py"*. bases its control law on the information given by the camera located below the robot. In the package pipefollowing[1] provided there are 3 scenarios and each of them has 2 possible environment behaviours: one kinematic and one dynamic. As evident, the dynamic scenarios are harder and the errors are higher in scale, while the error trend is similar.

2 Waypoints control

As mentioned in section 1 there are two possible environment behaviours: the kinematic one allows the robot to be governed by velocity inputs, while the dynamic one requires thrust inputs. It definitely matter when we plan the control. The transfer function of the robot for both cases are:

$$T(s) = \frac{P}{V} = \frac{P}{s\,P} = \frac{1}{s} \tag{1}$$

where P is the current position (output), V is the current velocity (input) and the derivative of P ($V = s\ P$).

$$T(s) = \frac{P}{F} = \frac{P}{M\ a} = \frac{P}{M\ (s^2\,P)} = \frac{1}{M\ s^2} \tag{2}$$

where P is the current position (output), F is the current thrust (input). the thrust is basically a force given to the vehicle and has this structure: $F = m\ a$. Where m is the mass of the vehicle, but, since it is working underwater, there are other parameters to take into account, like buoyancy and drag force. a is the acceleration and it is the double derivative of the position ($a = s^2\ P$). Looking at the definition of thrust it can be calculated as:

$$F = \frac{\rho\ A\ v^2}{2} \tag{3}$$

[1]Available at Javier Perez's repository: https://github.com/perezsolerj/pipefollowing

where ρ is the fluid density, A is the area of the fan and v is the velocity of the fluid pushed by the fan. This formula cannot be used in the transfer function definition because v is referred to the fluid while P and V re referred to the vehicle.

The figure 1 shows the control loop; the block called “SYS” is the vehicle, while the block called “K” is the controller. “P des”, “P err”, and “P act” mean “desired position”, “position error”, and “actual position” respectively. As we can understand our control is based on the position of the vehicle and the point to reach. According to this control loop the whole transfer function as this formula:

$$T(s) = \frac{G_c(s)\,G_s(s)}{1 + G_c(s)\,G_s(s)} \tag{4}$$

where $G_c(s)$ is the transfer function of the controller and $G_s(s)$ is the transfer function of the vehicle.

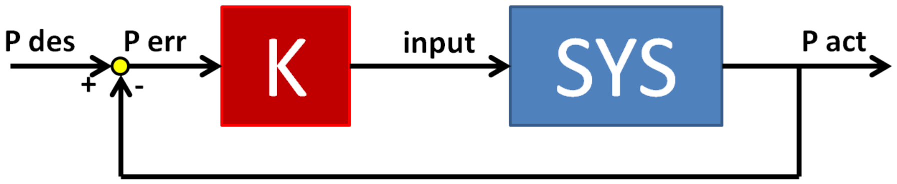


Figure 1: Control structure

The most popular control is the PID (Proportional - Integral - Derivative) without any doubt. However, we are going to use a P control for the kinematic environment and a PD control for the dynamic environment. The proportional part increases the input according the error magnitude, the integral part increases the input according to the sum of all the previous errors (useful to reach moving point), the derivative part decreases the input according to the error reduction (it helps to not overpass the point to reach). The transfer function of the PID controller has the following formula:

$$T(s) = K_p + \frac{1}{s}\,K_i + s\,K_d \tag{5}$$

Finally, the transfer function of the whole control loop for the kinematic environment is:

$$T(s) = \frac{K_p\,\frac{1}{s}}{1 + K_p\,\frac{1}{s}} = \frac{K_p}{s + K_p} \tag{6}$$

In order to guarantee the stability, all the poles must be negative; in this report the poles are imposed to be in -0.5. So:

$$s + 0.5 = s + K_p \quad => \quad K_p = 0.5 \tag{7}$$

The transfer function related to the dynamic environment is a little more complex:

$$T(s) = \frac{(K_p + s\,K_d)\,\frac{1}{M\,s^2}}{1 + (K_p + s\,K_d)\,\frac{1}{M\,s^2}} = \frac{K_p + s\,K_d}{s^2 + s\,\frac{K_d}{M} + \frac{K_p}{M}} \tag{8}$$

Since M is not easy to be defined, the analytic approach was not used. After some tests, $K_p = 0.4$ and $K_d = 1.6$ appeared to be the best values.

2.1 Algorithm principles

Since the vehicle can translate along the 3 main axes (X,Y,Z), the control can use a pure-translation strategy or a combined rotation-translation strategy. A combined strategy has been developed, but a pure-translation strategy has been analysed too. In particular pure-translation has been used to diminish the z-axis error, while rotation with translation has been used to diminish x-axis and y-axis error.

Before going thought the algorithm, I want underline the real robot has a maximum speed and so one has been imposed to simulated robot: it was 0.5 for both inputs (velocity, thrust). Moreover, we defined a minimum error. If the error is lower than a threshold the motors would turned off. The minimum error for the kinematic environment is 0.01 [m], while for the dynamic environment is 0.05 [m].

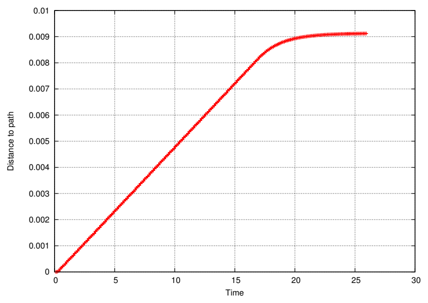
For both environments, the pure-translation works as follow: the velocity along every axis is imposed according to the distance between the robot and the point to be reached. If that value overpass the maximum velocity a saturation process is activated. The rotational velocity, instead, requires a specific approach for every environment kind: for the kinematic environment a simple proportional algorithm solves the problem, while the dynamic environment requires a little more complicated algorithm. The rotation is imposed according to the difference of the tangents of the angles, instead of the angles directly, because the arctangent may generate singularities. The dynamic environment, instead, requires almost no translation during the rotation. the robot starts to drift too fast and goes far from the right direction. So, when the robot has to turn, the velocity along the x-axis is imposed to null and the velocity along the y-axis is scaled down by 4. Initially, the control makes the robot turn into the point to reach, but it generates a lot of uncontrolled drifts. It drifted a lot for every turns because its priority was pointing to the next point and not staying close to the pipe. So, a new approach was defined: the robot has to align itself on the line connecting to point already passed and the point to reach. Moreover, the velocity along the x-axis is scaled down by 4 if the y-axis error is higher than the minimum error acceptable and the vehicle is not enough close to the point to reach.

In section 2.2 graphs about absolute error path and integrated squared error path are presented. As evident, the dynamic scenarios are harder and the errors are higher in scale, while the error trend is similar. The figures going from 2 to 5 are referred to the rotation-translation controller, which is the more realistic. The figures 6 and 7 are related to the comparison between the pure-translation and the rotation-translation strategy in the "dynamic turn scenario". As we can see the pure-translation strategy seems to be the better in terms of both time and precision, but it reduces the manipulation capabilities of the arm.

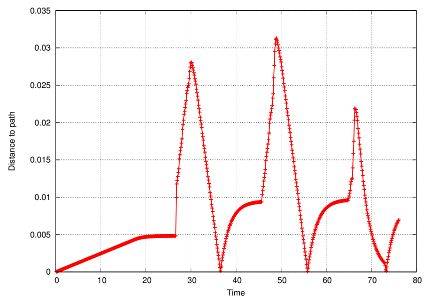
At the following hyper-links there are some control demo:

- Kinematic turn scenario - `https://www.youtube.com/watch?v=OOEzP_zwZv4`
- Dynamic turn scenario (pure-translation) - `https://www.youtube.com/watch?v=SCVOjyrzCZM`
- Dynamic turn scenario - `https://www.youtube.com/watch?v=Dw5mCuNHxgc`
- Dynamic height scenario - `https://www.youtube.com/watch?v=0EWqbmT82Eg`

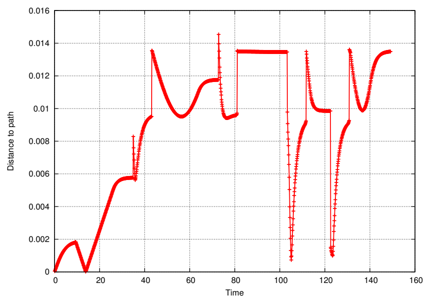
2.2 Graphs



(a) Basic

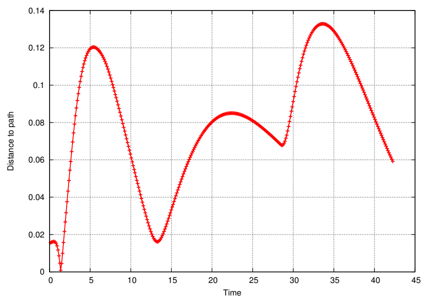


(b) Turns

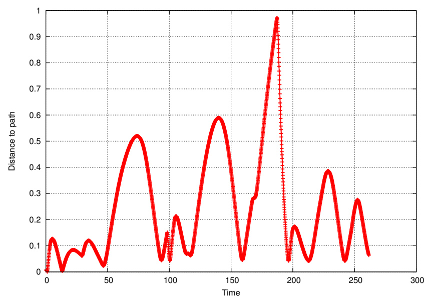


(c) Heights

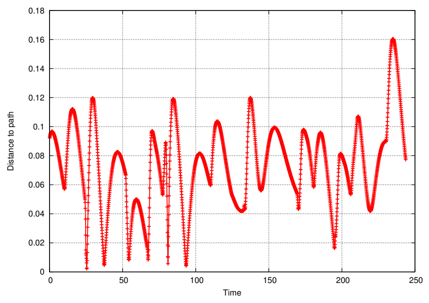
Figure 2: Waypoints - Error Kinematic Scenarios



(a) Basic

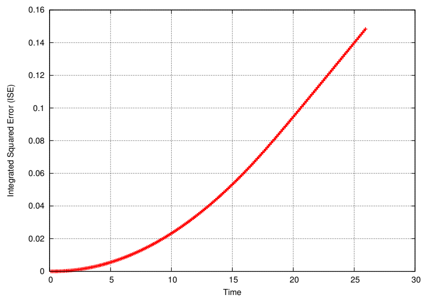


(b) Turns

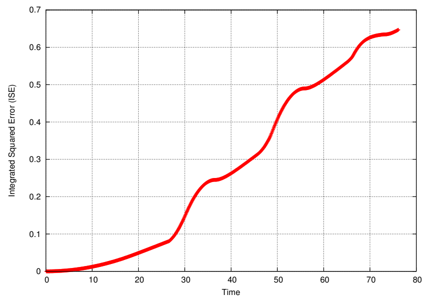


(c) Heights

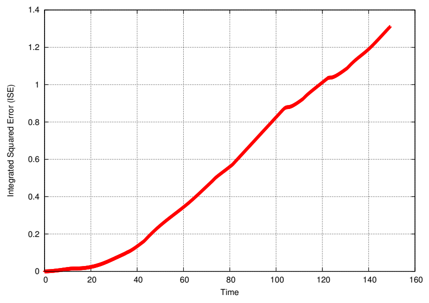
Figure 3: Waypoints - Error Dynamic Scenarios



(a) Basic

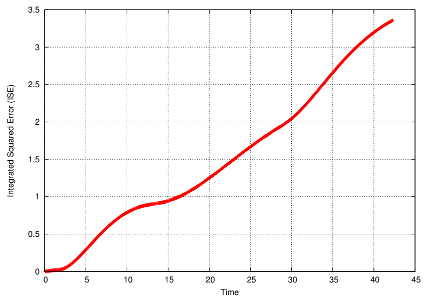


(b) Turns

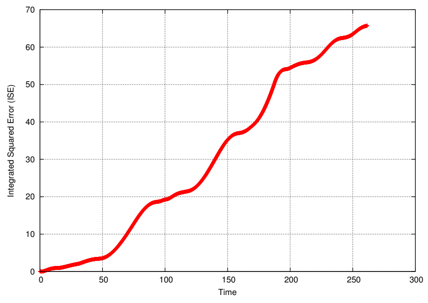


(c) Heights

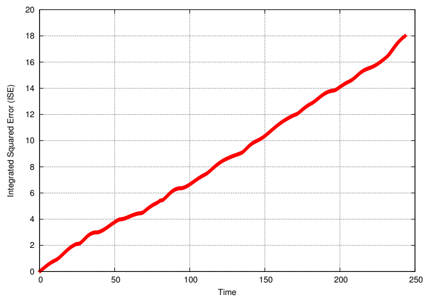
Figure 4: Waypoints - ISE Kinematic Scenarios



(a) Basic

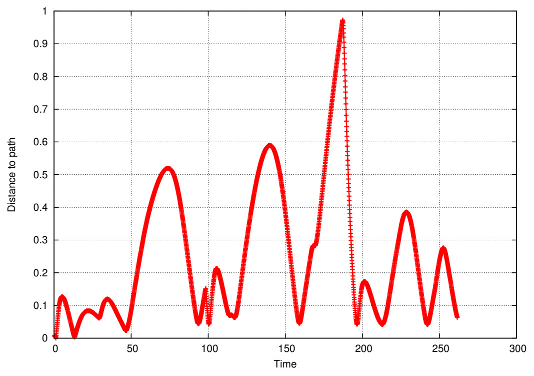


(b) Turns

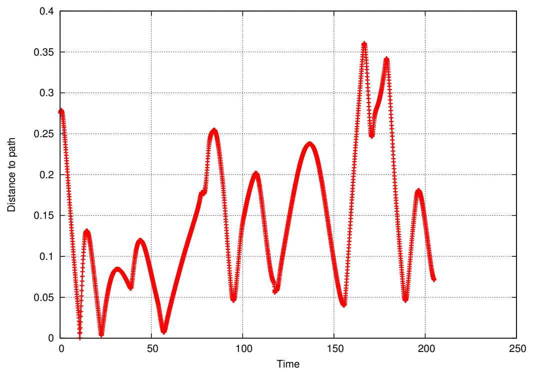


(c) Heights

Figure 5: Waypoints - ISE Dynamic Scenarios

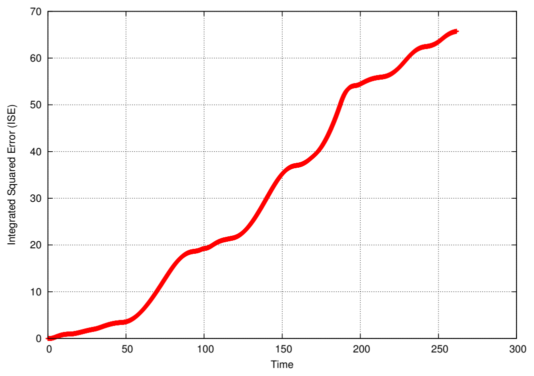


(a) Rotation-translation strategy

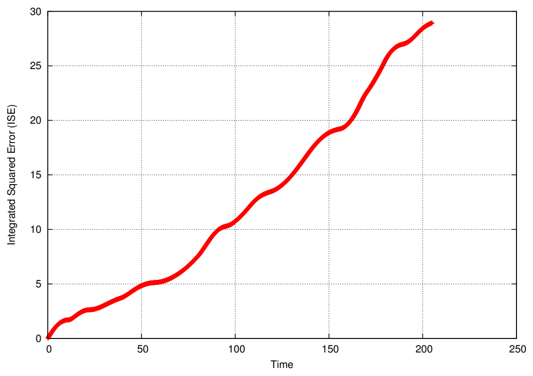


(b) Pure-translation strategy

Figure 6: Waypoints - Error Dynamic Turn Scenarios



(a) Rotation-translation strategy



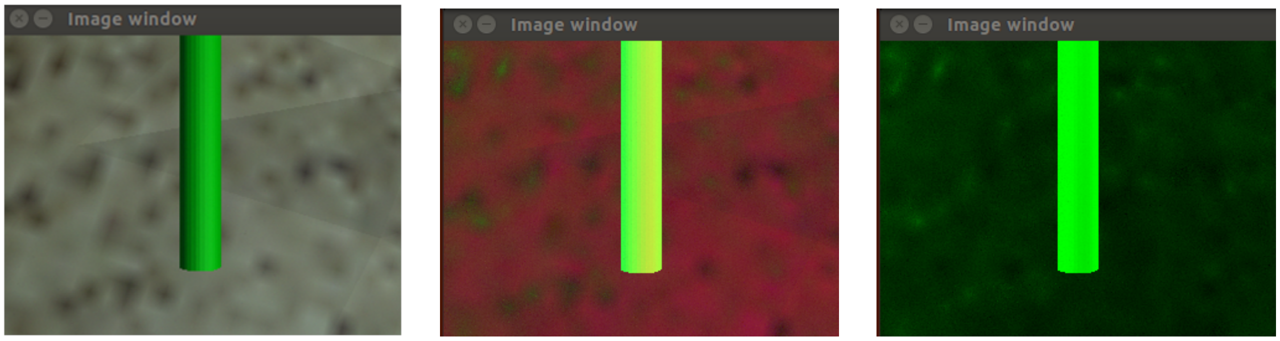
(b) Pure-translation strategy

Figure 7: Waypoints - ISE Dynamic Turn Scenarios

3 Vision Control

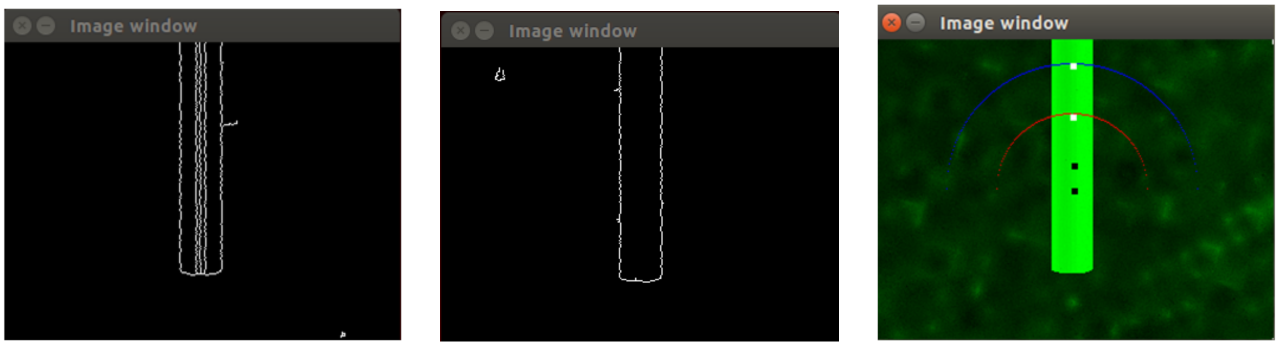
As mentioned in section 1, this control bases its algorithm on the camera information. The original image acquired by the camera is an RGB image (fig. 8(a)); using directly a *"Canny"* function to extract the borders is pretty dangerous. This function sees shades and shadows as borders (fig. 9(a)). This image helps to se the direction of the pipe, but it does not helps much to see the thickness of the pipe. Without knowing the thickness of the pipe we cannot establish the distance vehicle-pipe. A pre-image process is definitely required. The RGB image has been transformed to an HSV image (fig. 8(b)) and then just the green layer kept (fig. 8(c)). Finally, the *"Canny"* function gave a useful results (fig. 9(b)).

In order to extract all the information required to make the vehicle span the pipe, two half circumferences are laid on the image (fig. 9(c)) and the intersection with the *"Canny"* image evaluated. The two circumferences have 60 and 100 pixel has radius. The two black dots located in the centre of the figure 9(c) represent the vehicle and its orientation. The two white dots represent the averaged points of all the intersections of every half-circumference. Sometimes, the curve intersects in more than two points due to pipe shape, or *"Canny"* function errors. If the intersected points are not exactly two, the location of the averaged points are not updated.



(a) Original (b) HSV (c) Green Layer

Figure 8: Vision - Image Colour Processing



(a) Canny from the original picture (b) Canny from the green layer (c) Definitive processed image

Figure 9: Vision - Image Canny Processing

The distance between the two white points gives the position error along the x-axis, the angular coefficient (tangent of the angle) of the line passing through the white points gives the angular error, the difference between the y values of the white point and the black points gives the position error along the y-axis, and the distance between the intersection circumferences-pipe boards represents the height. So, if it is bigger than the established one (33 pixel), the control will push the vehicle up, otherwise down.

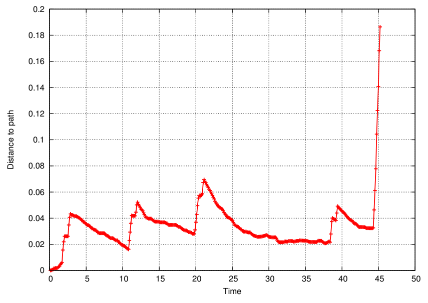
As for the waypoints control (sec. 2) we use two different controller: a P for the kinematic environment and a PD for the dynamic environment.

In section 3.1 graphs about absolute error path and integrated squared error path are presented. As evident, the dynamic scenarios are harder and the errors are higher in scale, while the error trend is similar. The figures going from 10 to 13 are referred to the rotation-translation controller.

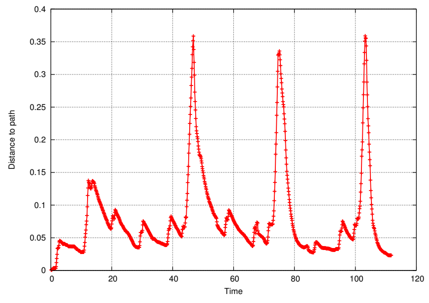
At the following hyper-links there are some control demo:

- Kinematic turn scenario - `https://www.youtube.com/watch?v=YkbFcK_LsW8`
- Kinematic height scenario - `https://www.youtube.com/watch?v=p5I0n-8OmCE&t=2s`
- Dynamic turn scenario - `https://www.youtube.com/watch?v=ZSFl2YIcmR0`
- Dynamic height scenario - `https://www.youtube.com/watch?v=KGwu1OKH44Q`

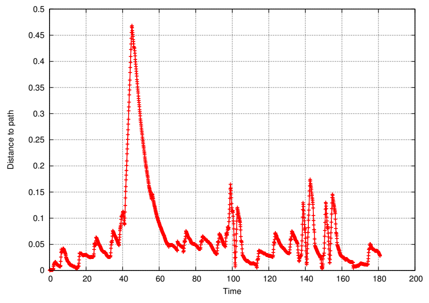
3.1 Graphs



(a) Basic

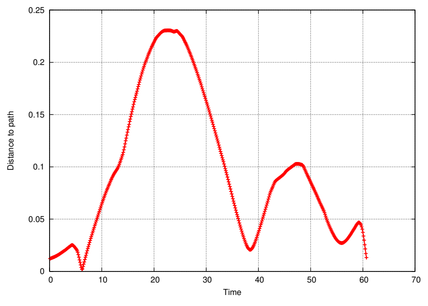


(b) Turns

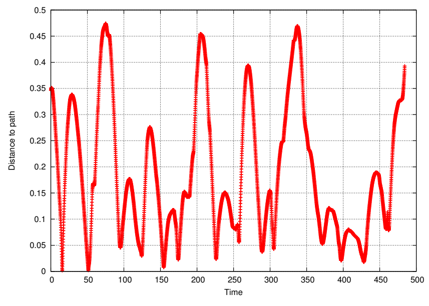


(c) Heights

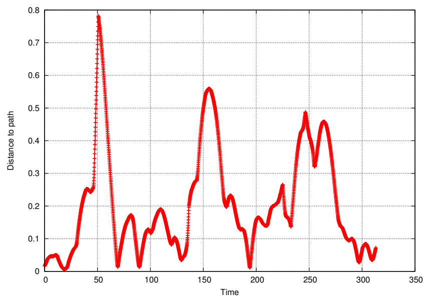
Figure 10: Vision - Error Kinematic Scenarios



(a) Basic

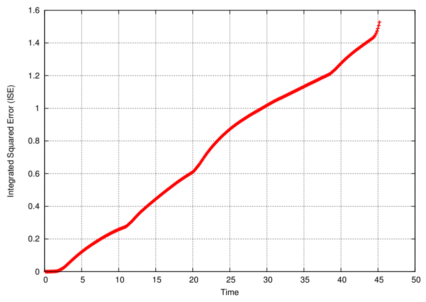


(b) Turns

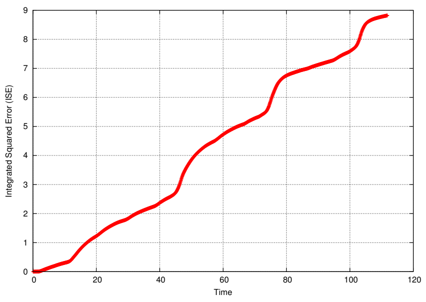


(c) Heights

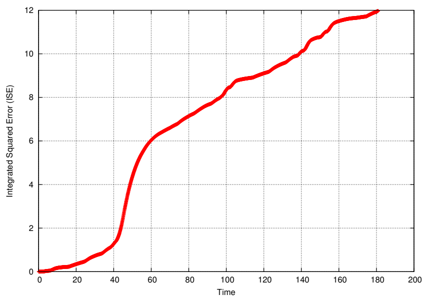
Figure 11: Vision - Error Dynamic Scenarios



(a) Basic

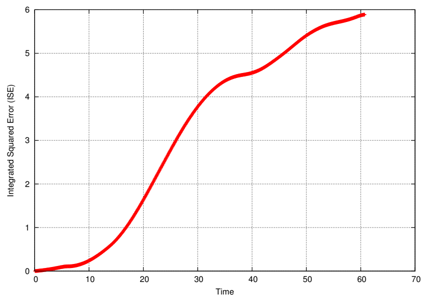


(b) Turns

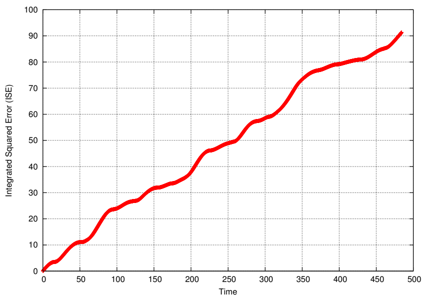


(c) Heights

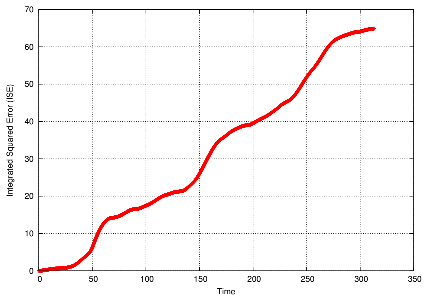
Figure 12: Vision - ISE Kinematic Scenarios



(a) Basic



(b) Turns



(c) Heights

Figure 13: Vision - ISE Dynamic Scenarios

References

[1] Lessons and notes of prof. José V. Martí, James I University

[2] Wiki.ros.org. (2016). uwsim - ROS Wiki. [online] Available at: http://wiki.ros.org/uwsim

[3] Ribas, D., Palomeras, N., Ridao, P., Carreras, M. and Mallios, A. (2012). Girona 500 AUV: From Survey to Intervention. IEEE/ASME Transactions on Mechatronics, 17(1), pp.46-53.