Master in Computer Vision

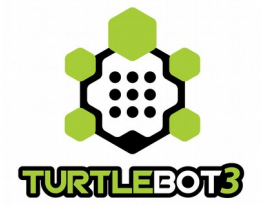
UNIVERSITE DE BOURGOGNE

Visual Servoing

Technical Report

Supervisor : Ph.D. OMAR TAHRI

Student: Martin Emile



Final Report for the course titled “Visual Servoing”

Centre Universitaire Condorcet - UB, Le Creusot

December 27, 2020

Contents*

1-Abstract

Demonstration of Visual Servoing project is developed under ROS environment and using TheConstruct platform with TurtleBot running on Gazebo web version "Gzweb". The main idea of the project is to develop an application running on TurtleBot to solve different navigation problems. This project can be sub-divided into 2 parts, Line following task and Parking task. For each task, control and simulation is achieved through different implementations, which in order to be solved, a certain skills related to ROS basics and their programming architecture understanding should be mastered .On top of that, Our simulated ROS environment provided by The-Construct website is exploited with different ROS online courses. In this report, a detailed guideline about the implementation will be presented in order to deliver the key features of our approach.

1- Introduction

- Visual servoing refers to closed loop position control of a robot end-effector, using measurements from a vision sensor as mentioned in [3]. Two kinds of configurations for visual servo control are most frequently encountered in literature:

1-Eye-in-hand

2-Eye-to-hand

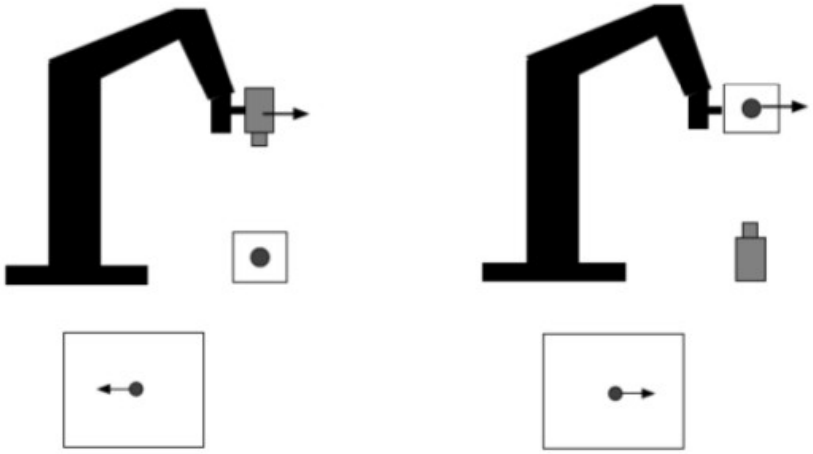


Figure 1: Eye-in-hand (left) and Eye-to-hand (right) configurations in visual servoing [3]

- TurtleBot3 is a one low-cost, personal robot with open-source software, that can perform different interesting goals. According to [1], TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education and research. The goal of TurtleBot is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expand-ability. The TurtleBot3's core technology are Navigation and Manipulation, making it suitable for home service robots. The TurtleBot can run SLAM(simultaneous localization and mapping) algorithms to build a map and can drive around the environment. Also, it can be controlled remotely from a laptop, joy-pad or Android-based smart phone as described in [2]. The TurtleBot can also follow a person's legs as they walk in a room.

- Our goal is to implement Two tasks on Turtlebot3 robot, that is based on mainly on knowing how to use ROS to manipulate the robot, simulate an environment, as well as integration of the library OpenCV along with ROS

Task 1: Line following robot
Task 2: Parking Robot

2-Result Demo Videos: **Demo Link** , **Github Link :Github Repo**

3-Problem approach

For task achieving, The-Construct platform that has all the necessary ROS packages was chosen for implementation of the two tasks.

A review of Linux basic shell commands, such as, How to navigate through a Linux file system. How to interact with a Linux file system, How to edit files using the Shell (vi editor). Manage access to files (Permissions).Create simple Linux programs (Bash Scripts). Manage execution of Linux programs (Processes).How to connect to the remote computer of a robot (ssh). Were covered.

Visual servoing Project Implementation

Project Progress pipeline

COMPLETED

1st Revision
-Linux
-Python

2nd Revision
-Ros in 5 days

OpenCV
-Integration with ROS

Gazebo
-Spawn enviroment

Map color
-HSV to mask to centroide computation

Tracking
-Track the computed centroide

Review
-How to spawn models in gazebo

Generate AR
-Generate and spawn an ar_tag

Track tag
-Use ar_track_alvar package

Intro
-Cover documentation

Spawn env.
-Run Core launch file

Test&Debug
-In Progress

PROJECT START

15/11 16/11 - 18/11 20/11 - 25/11 28/11 - 8/12 13/12 - 27/12

Review
Get the neccessary knowledge

Task 1
Line-Following

Task 2 Part 1
AR-TAG

Task 2 Part 2
AutoRace

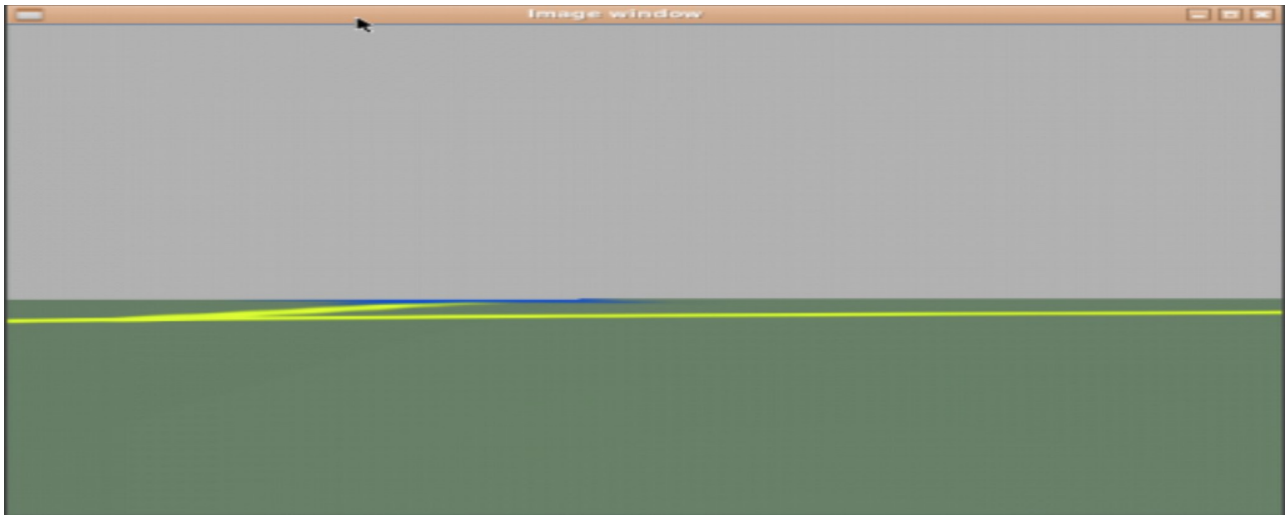


Figure (2).

5- Task 1 approach:

•**Definition:** script that moves the robot to follow a line on the environment

• **Pre-acquisitions:** Most important step: How to define a centroide or a point the robot can follow that can change its location based on the color of the line

• **Fundamental Pipeline:**

1- Create a ROS package with dependency in rospy

2- Create a python file that subscribe to the topic “/camera/rgb/image_raw"

3- Get Images from this topic and show them with OpenCV Using **OpenCV_bridge.** This package allows the ROS imaging topics to use the OpenCV image variable format.This program will give you an image similar to figure (2).

4- Cropping the input image. It's very important to work with the minimum size of the image required for the task. This makes the detecting system mush faster.

5- Convert from BGR to HSV:The idea behind HSV is to remove the component of color saturation. This way, it's easier to recognise the same color in different light conditions, which is a serious issue in image recognition. As mentioned in [4]

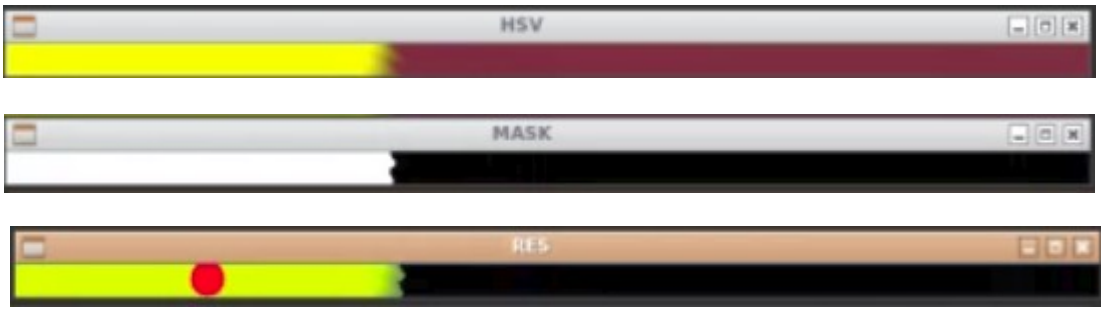


Figure (3). from top to bottom, (A)HSV image, (B) corresponding mask, (C)computed centroid in red circle.

6- Apply the mask: generating a version of the cropped image in which we only see two colors: black and white. The white will be all the colors we consider yellow and the rest will be black. It's a binary image.see the above figure(3).

7-Get The Centroids, draw a circle where the centroid is and show all the images: The place where there is more of the color that you are looking for is where the centroid will be. It's the center of mass of the blobs seen in an image. See Figure (3) (C).

8-Move the TurtleBot based on the position of the Centroid: this control is passed on a Proportional control, and it will always gives a constant linear movement and the angular Z velocity depends on the difference between the centroid center in X and the center of the image.

6- Task 2 – 1st approach:

•**Definition:** script that moves the robot towards a generated AR_TAG using the tracking package ar_track_alvar

• **Per-acquisitions:** How to spawn a customized AR_TAG in gazebo

• **Fundamental Pipeline:**

1- Create a ROS package with dependency in rospy

2- Get info from the package ar_track_alvar to generate Tags

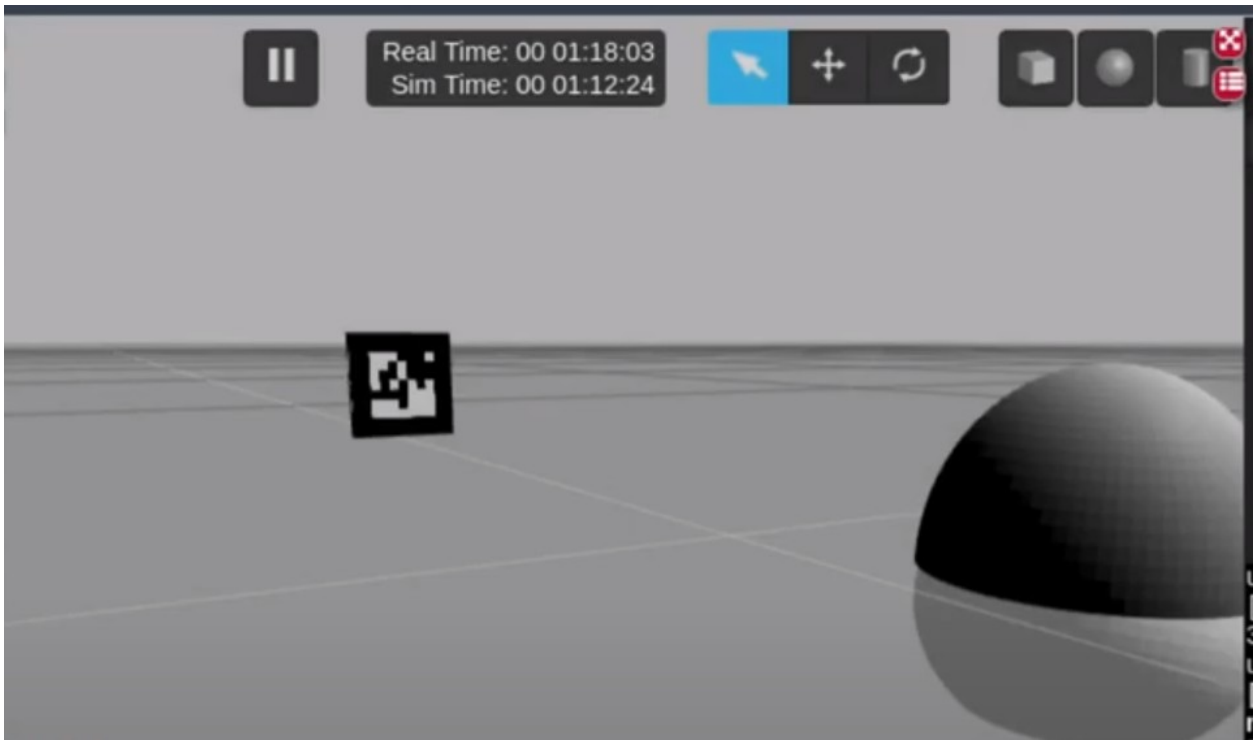


Figure (4).

3- **ar_track_alvar:** has 4 main functionalities we are only concerned with the following two:

- Generating AR tags of varying size, resolution, and data/ID encoding

-Identifying and tracking the pose of individual AR tags, optionally integrating kinect depth data (when a kinect is available) for better pose estimates, as described in [5].

4- Generating the Tag with ID=1, as in figure (4), using the commend **“`rosrun ar_track_alvar createMarker`”**

5- Now, we need to create a collada file (.dae extention) to spawn the tag into gazebo, to do that we used “ blender” software to stretch the generated tag as a texture along a cuboid with dimension (9x9x1) cm.

6- Create a package with rospy dep. And upload the generated collada file into a model directory.

7- Create a launch file to spawn the model into gazebo along with the ” .urdf file”

8- The next step is to make an implementation with the library to track the generated tag and to publish into “/cmd_vel” with the right position in order for the robot to park

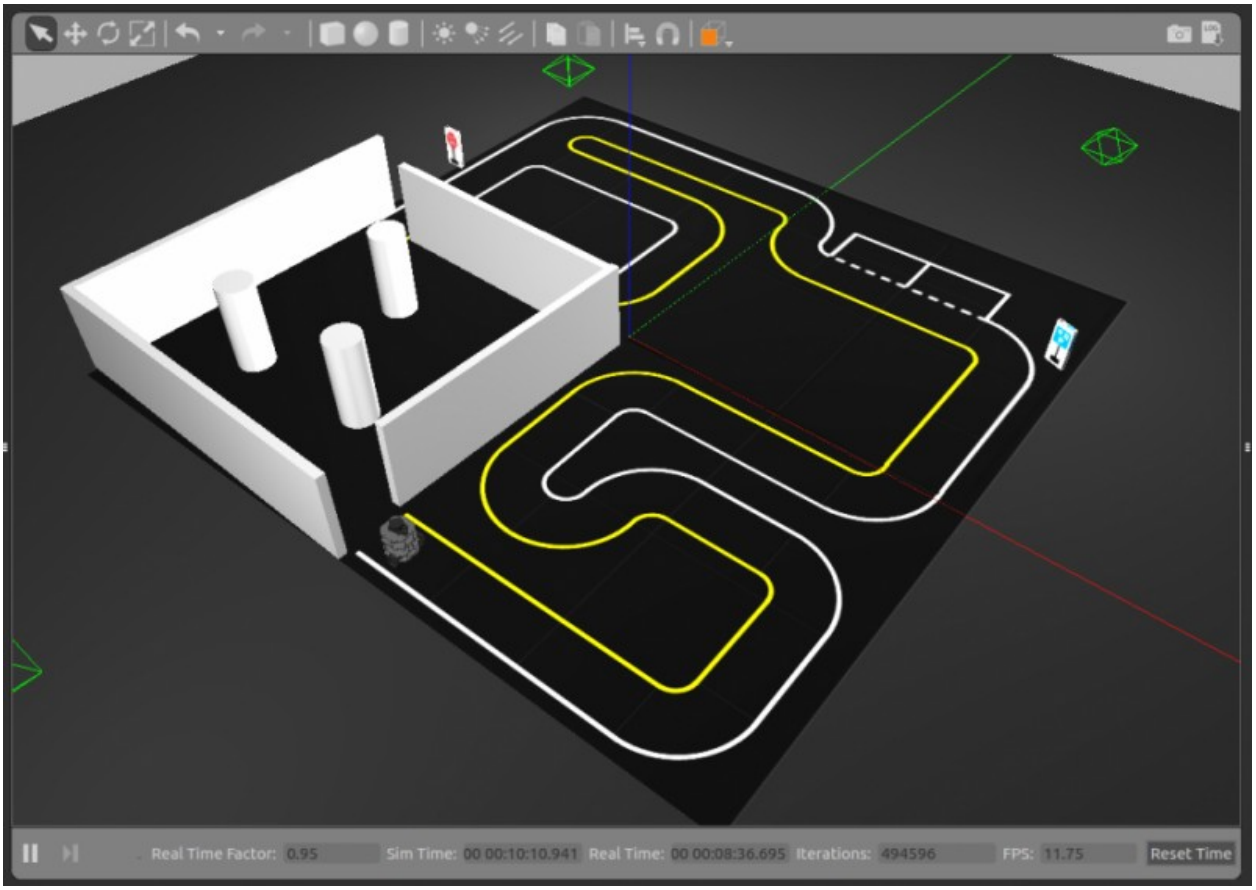


Figure (5) Auto-race Track

Since I did not manage to solve the parking task through the first approach, I had a second approach using the auto-race track for TurleBot3

7- Task 2 – 2nd approach:

•**Definition:** Script that moves the robot in the auto-race track for TurtleBot3,

• **Pre-acquisitions:** Clone the auto-race package with all the files for the different tasks along side with the parking task.

• **Fundamental Pipeline:**

1- Clone the github repo of the auto-race for turtlebot3

2- Launch the gazebo launch file for the track using “roslaunch turtlebot3_gazebo turtlebot3_autorace.launch”
3- Launch the mission file using “roslaunch turtlebot3_gazebo turtlebot3_autorace_mission.launch”

Figure (6). Parking sign image taken from the simulation Gazebo then resized

4- Asign some important environment variables, such as :

-export AUTO_IN_CALIB=action
-export GAZEBO_MODE=true
-export AUTO_EX_CALIB=action
-export AUTO_DT_CALIB=action
-export TURTLEBOT3_MODEL=burger

5- Calibration for intrinsic and extrinsic parameters using the following command:

-"roslaunch turtlebot3_autorace_camera turtlebot3_intrinsic_camera_calibration.launch"

-"roslaunch turtlebot3_autorace_camera turtlebot3_exitrinsic_camera_calibration.launch"

6- Run the lane keeping launch file in order for us to observe the parking sign in the simulation using rqt_image_view

7- Take a screenshoot of the parking sign as well as crop and resize it, finally, put in the directory for the algorithm SIFT

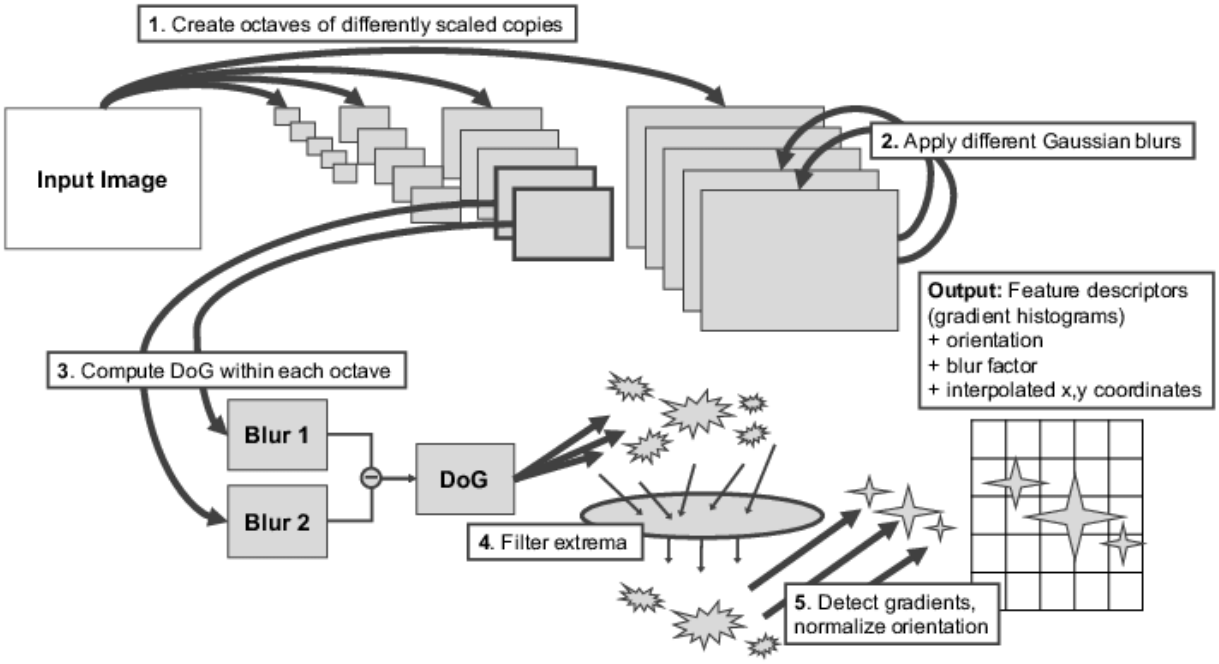
8-Launch the core file to perform all tasks in the autorace track using "roslaunch turtlebot3_autorace_core turtlebot3_autorace_core.launch"

9- publish an integer =2 to the topic called "/core/decided_mode using :

"rostopic pub -1 /core/decided_mode std_msgs/UInt8 "data:2"

we will find the robot perform the following tasks

1- Lane keeping or lane following
2- Traffic sign detection



3-Automatic parking

Figure (5). SIFT algorithm

The automatic parking algorithm is based on The scale-invariant feature transform **(SIFT)**

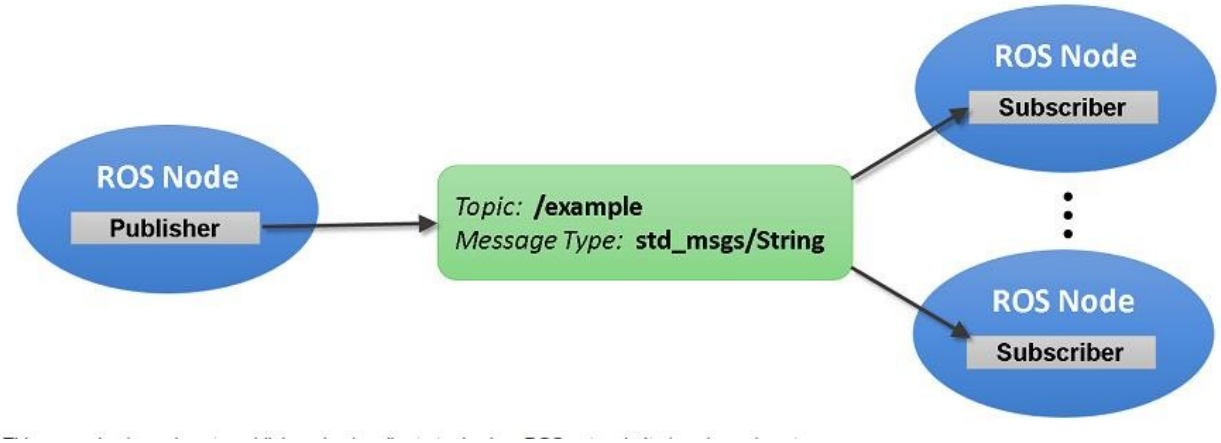
8-Why SIFT ?

-Locality: features are local, so robust to occlusion and clutter (no prior segmentation)
-Distinctiveness: individual features can be matched to a large database of objects
-Quantity: many features can be generated for even small objects
-Efficiency: close to real-time performance
-Extensibility: can easily be extended to a wide range of different feature types, with each adding robustness

The algorithm:

SIFT is quite an involved algorithm. There are mainly four steps involved in the SIFT algorithm.:

Scale-space peak selection: Potential location for finding features.
Keypoint Localization: Accurately locating the feature keypoints.



Orientation Assignment: Assigning orientation to keypoints.
Keypoint descriptor: Describing the keypoints as a high dimensional vector.
Keypoint Matching

Figure (6)

9- ROS Review & analysis:

A) Overview on the project:

Software: Using the environment on "The construct" platform with the fundamental ROS packages already installed in order to implement, debug and visualize our result.

C) ROS Fundamentals:
• How to structure and launch ROS programs (packages and launch files)

-ROS uses packages to organize its program, and all those files in the package are organized with the following structure:
- launch folder: Contains launch files
- src folder: Source files (cpp, python)
- CMakeLists.txt: List of cmake rules for compilation
- package.xml: Package information and dependencies• How to create basic ROS programs (Python based):
- Every Python ROS Node will have a declaration at the top, just to make sure your script is executed as a Python script.
For instance: "#!/usr/bin/env python"

•ROS Topics:
Topics are basically like a highway road in order for nodes to exchange messages, on top of that, Topics have anonymous publish/subscribe semantics. As in figure (6).

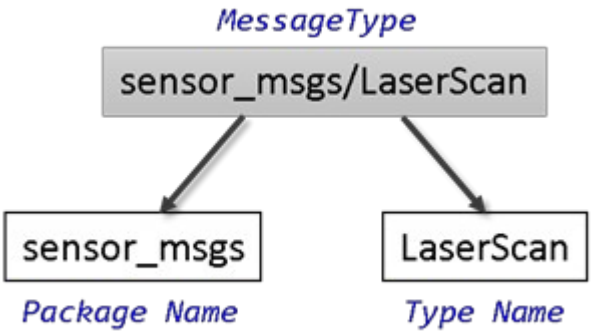


Figure (7).

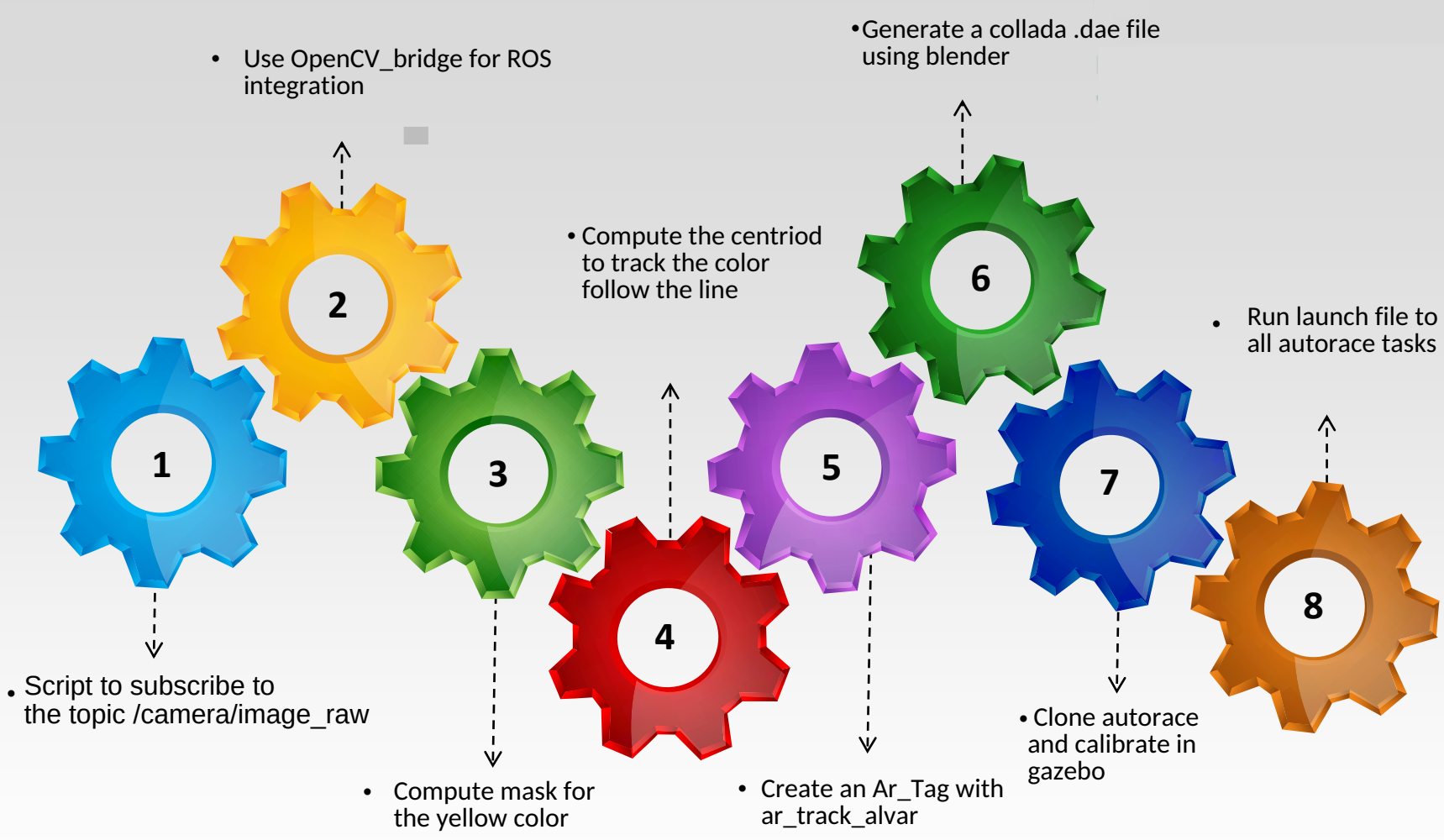
• **ROS Nodes:**

Basically, nodes are processes that perform some computation or task, A node can independently execute code to perform its task but can also communicate with other nodes by sending or receiving messages. The messages can consist of data, commands, or other information necessary for the application.

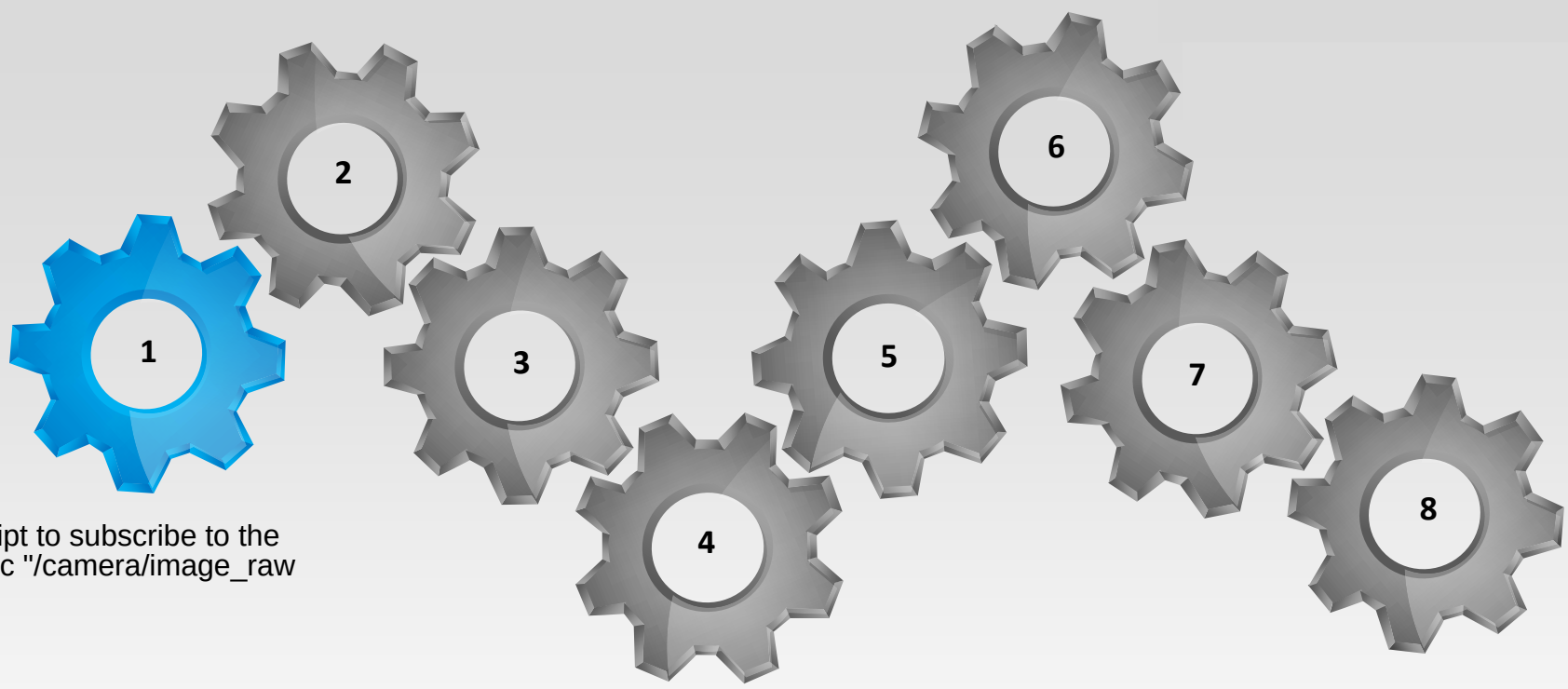
• **ROS messages:**

ROS messages are defined by the type of message and the data format. The ROS package named std_msgs, for example, has messages of type String which consist of a string of characters. Other message packages for ROS have messages used for robot navigation or robotic sensors. The turtlesim package has its own set of messages that pertain to the simulation. As in figure (7).

Project Process – 8 Stages to Visual servoing project



Project Process – 1st Stage

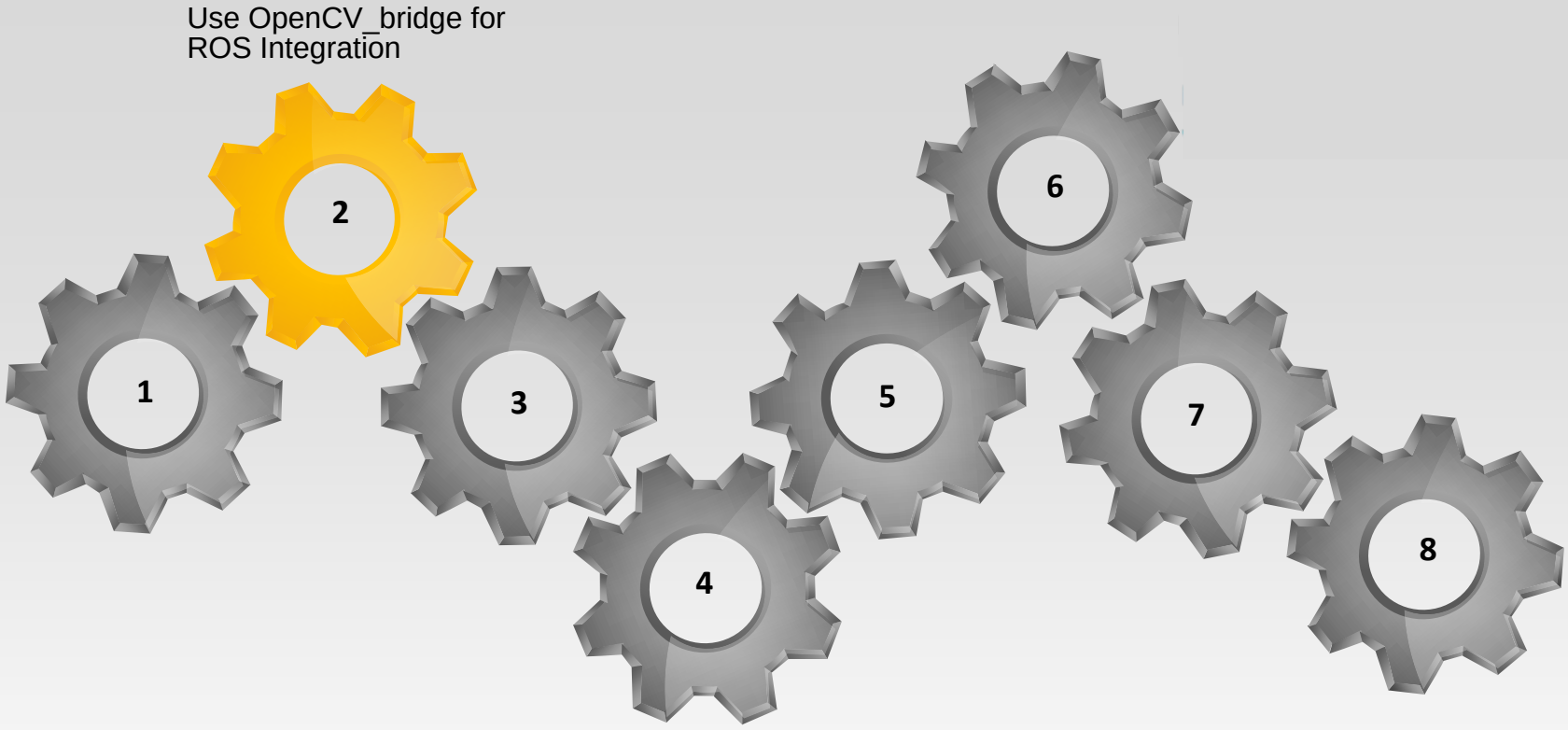


Script to subscribe to the topic "/camera/image_raw

1st Stage

Create a package with rospy dependancy to place the necessary fills
Get info with “rostopic info /camera/image_raw”
, after this,get info of the message type that the topic use. So, this is the type of message that we will need to subscribe to the topic in order to get the stream of images.

Project Process – 2nd Stage

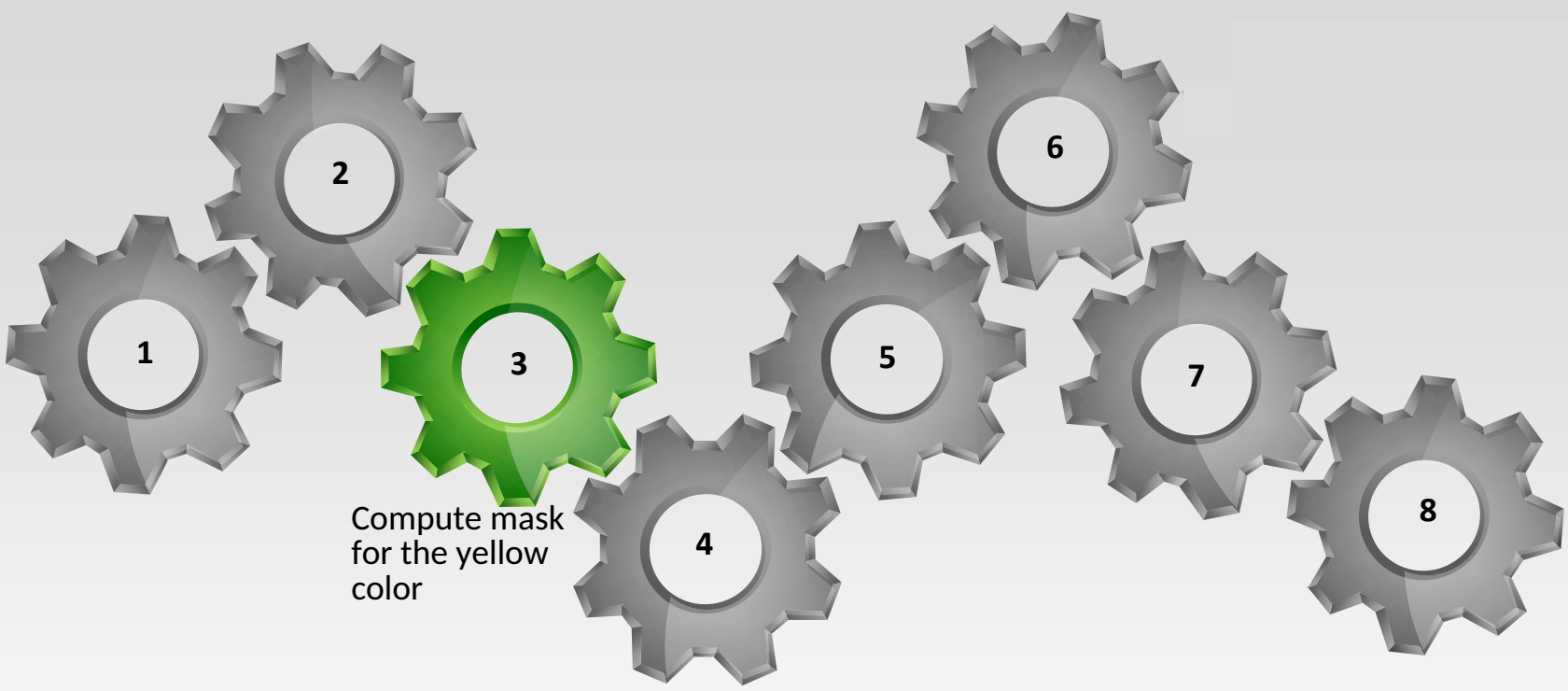


2nd stage

ROS passes around images in its own sensor_msgs/Image message format

CvBridge can be found in the cv_bridge package in the vision_opencv stack.

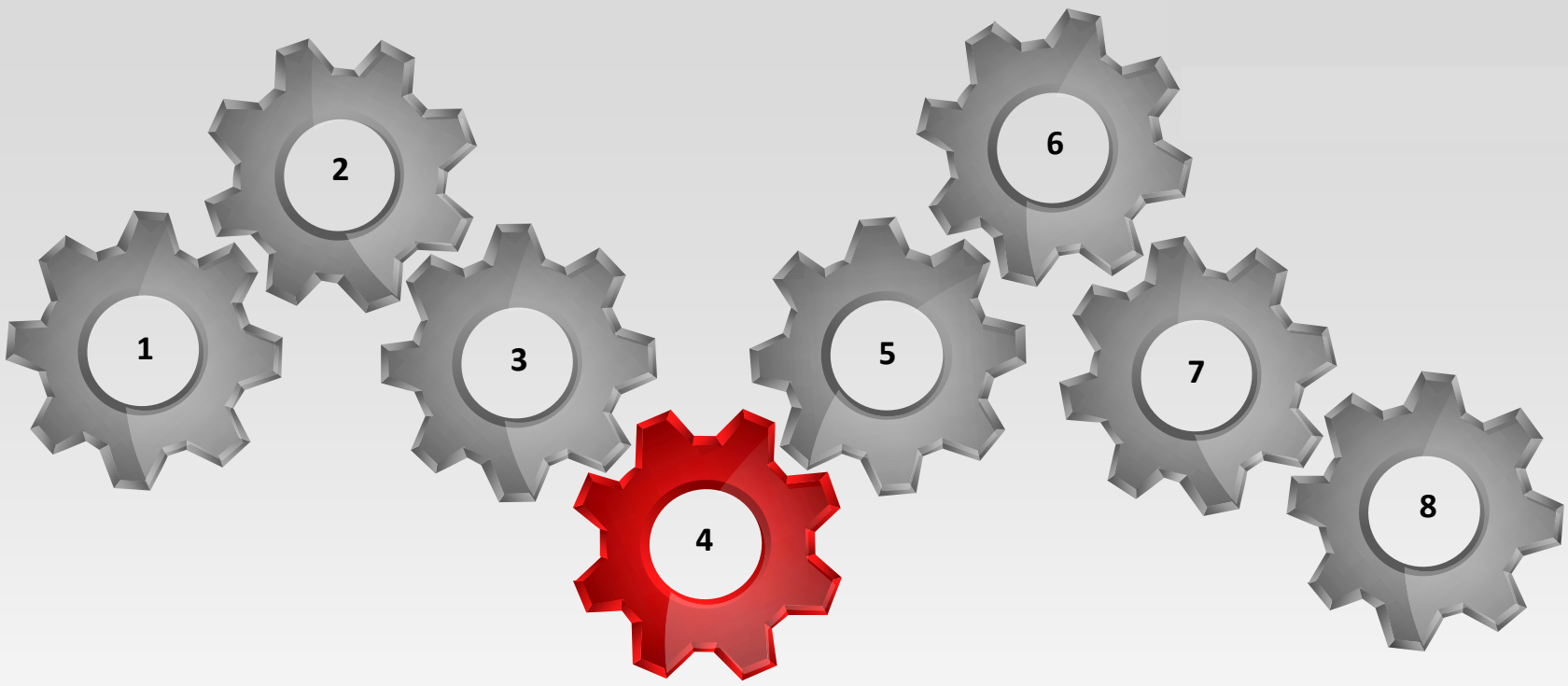
project process – 3rd Stage



3rd Stage

We need to convert the color to HSV so that the variation in the color intensity will not affect our detected color, then we get a threshold for the yellow color to have a mask

project process – 4th Stage

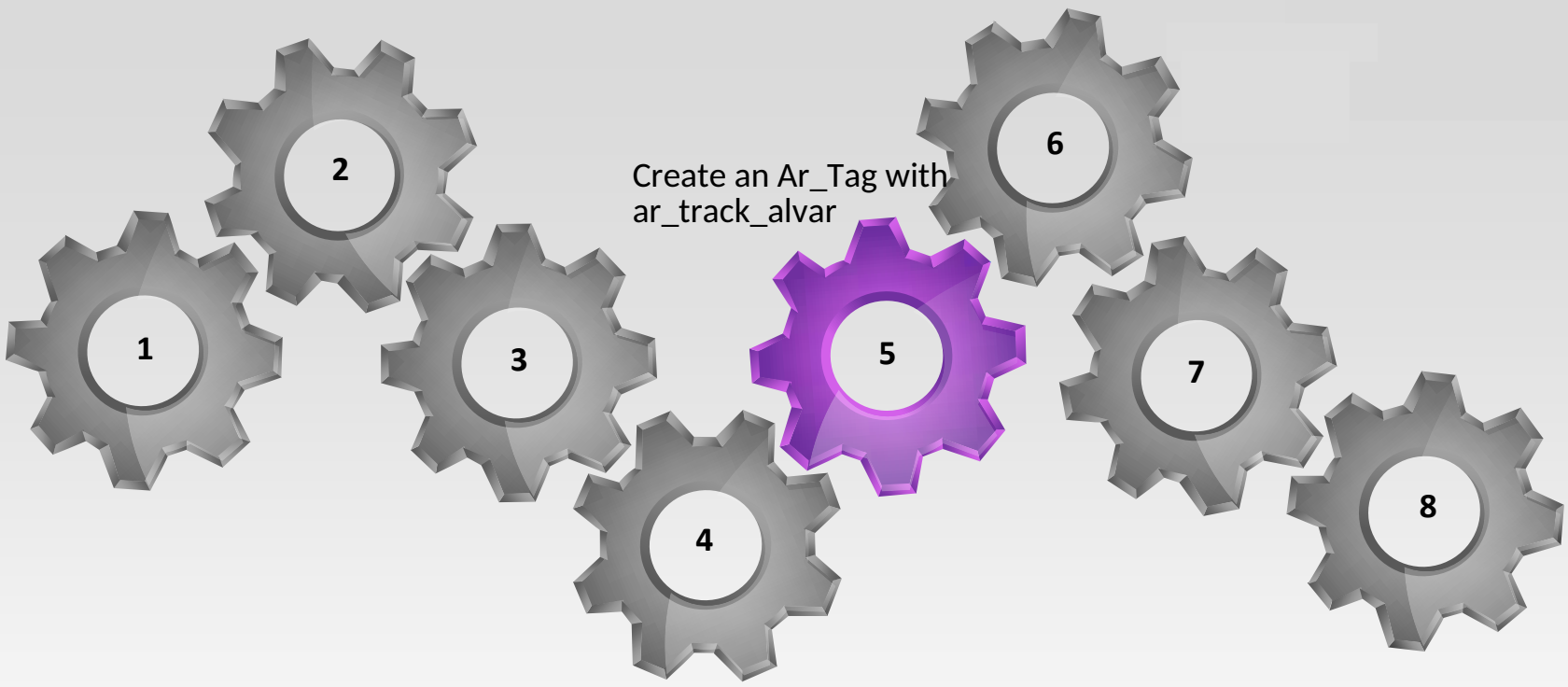


Compute the centroid to track the color
follow the line

4th Stage

In order for us to track a point, we need to compute the center "Centroid" of all the white pixels in our result mask, this will give us a point to track so that we apply movment to the wheels by publishing into "/cmd_vel" topic

project process – 5th Stage

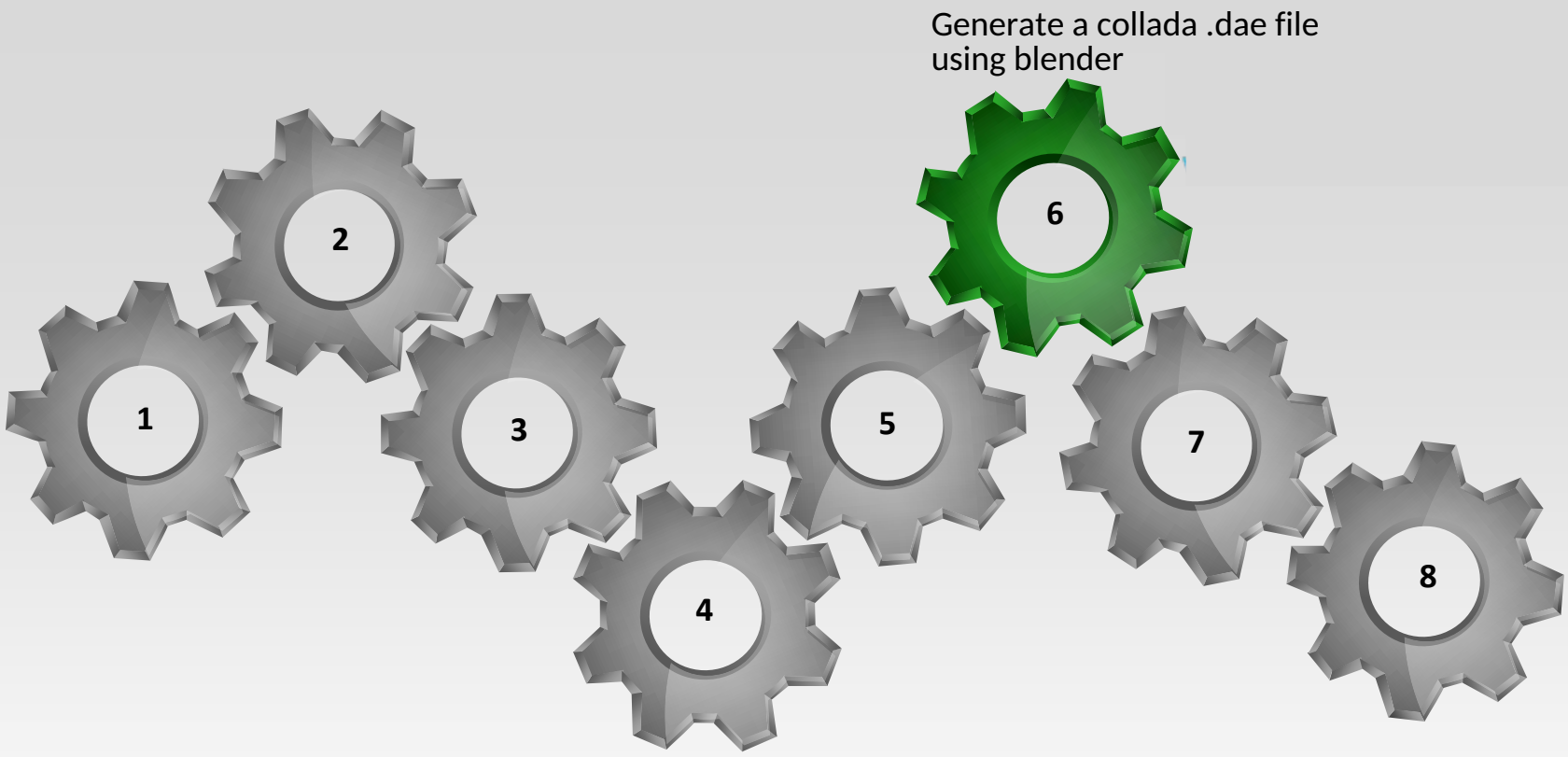


Create an Ar_Tag with
ar_track_alvar

5th Stage

Create a package with rospy dependency to create a new launch file in order to spawn the generated URDF model, then we use ar_track_alvar to generate a tag image

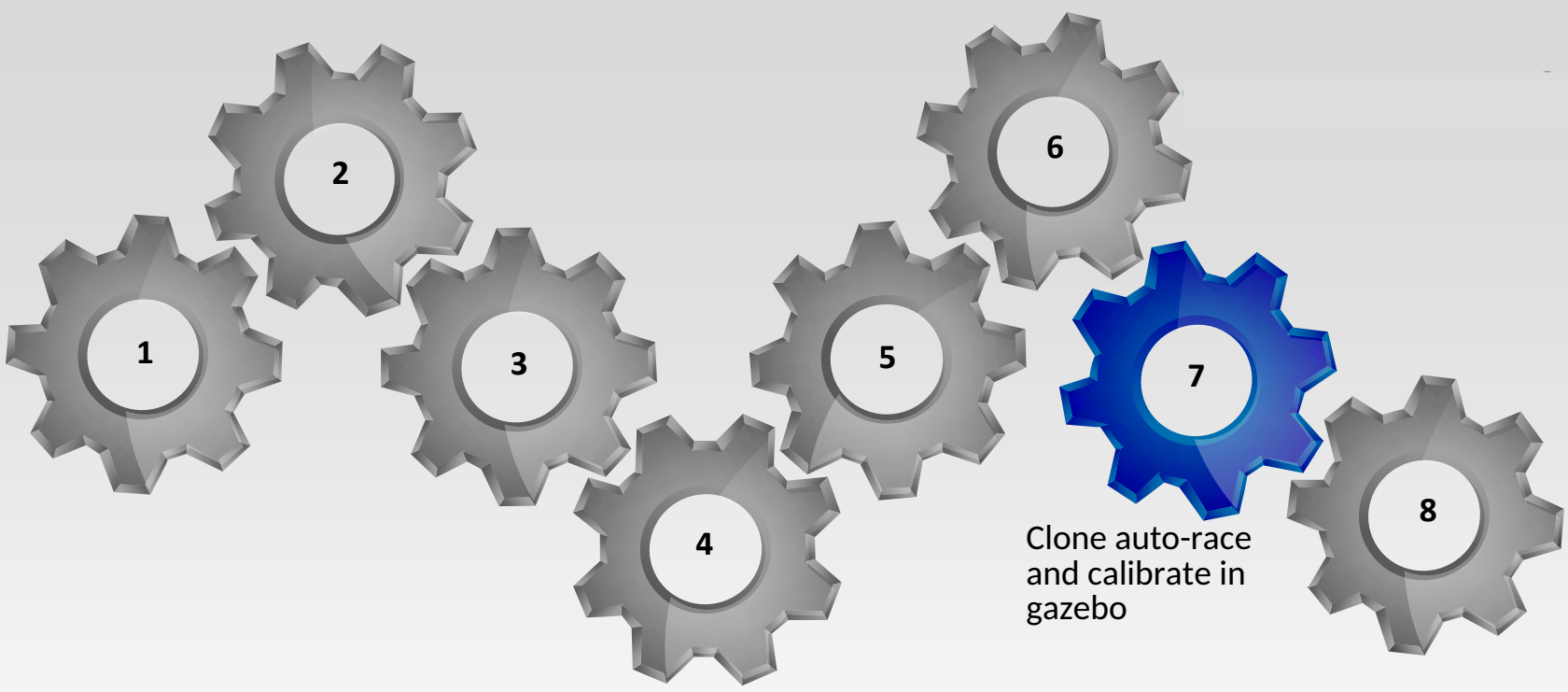
project process – 6th Stage



6th Stage

Using blender software we will wrap the generated Tag into a cuboid of dim(9x9x1)cm then to generate a collada file then into URDF file extension, then we create a launch file to spawn the URDF model into gazebo

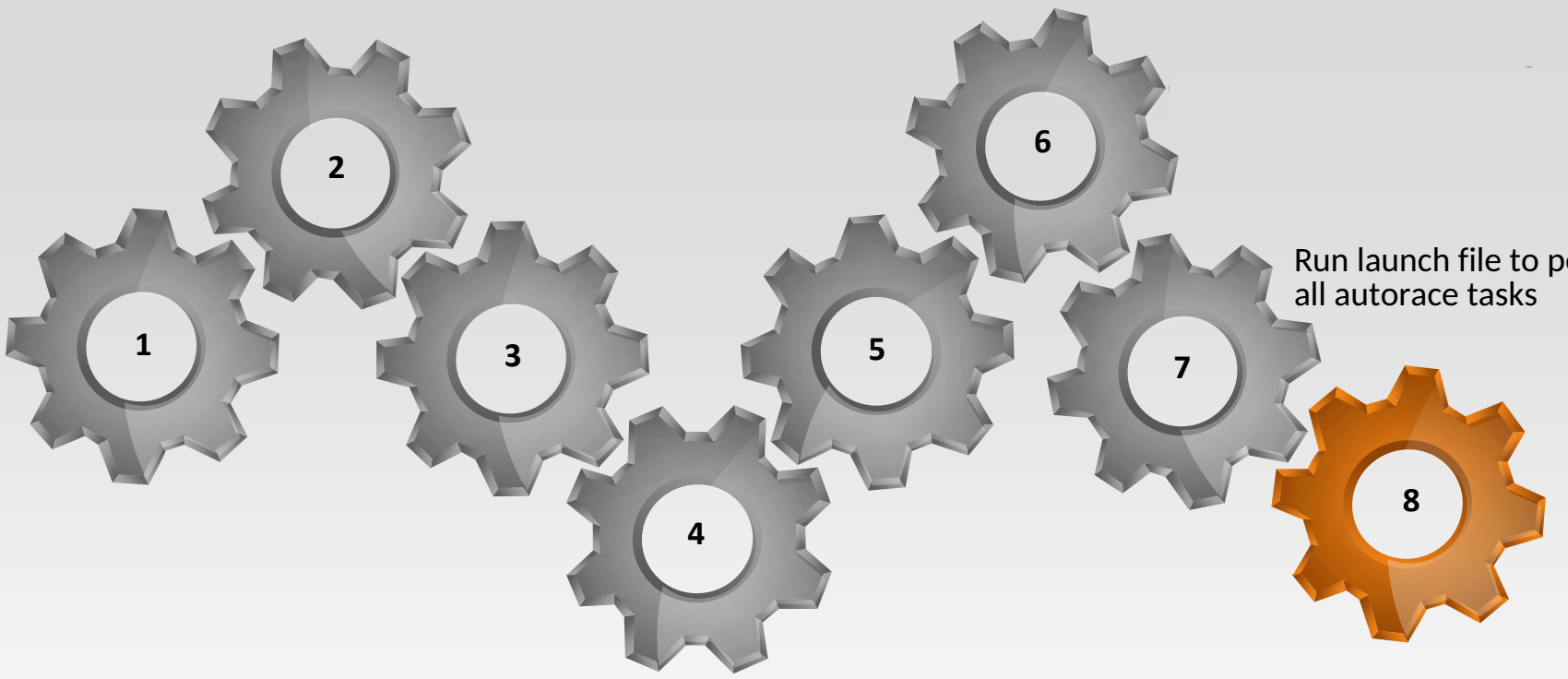
project process – 7th Stage



7th Stage

After cloning the auto-race repository, we need to execute two launch files one for calibration of the intrinsics and the other for the extrinsic parameters then we assign some necessary enviroment variables like the GAZEBO_MODE, TURTLEBOT3_MODEL .

project process – 8th Stage



8th Stage

We need to launch the core file for all the auto race tasks after we have done the calibration step, then we need to publish into the topic "/core/decided_mode"

12-Conclusion:

As a part of the visual servoing project, a demonstration of each task was covered, for the line-following part, a description of the pipeline of the chosen algorithm has been explained. on top of that, and for the parking task, we had two approaches to present since the first approach did not solve the problem, the first approach is the ar_tag tracking, while the second approach was to clone an already designed environment for the turtlebot3 named auto race, where we can perform differently tasks included automatic parking

13-References:

[1] https://emanual.robotis.com/docs/en/platform/turtlebot3/autonomous_driving_autorace/

[2] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[3] Visual servoing, Francois Chaumette, Inria, Rennes, France In Computer Vision: a Reference Guide, K. Ikeuchi

[4] Course titled “ROS Navigation in 5 days”

[5] Course titled “Mastering with ROS: Turtlebot3”

[6] Course titled “ROS basics in 5 days”

[7] http://wiki.ros.org/ROS/Tutorials/ExaminingPublisherSubscriber

[8] http://wiki.ros.org/navigation