University of Burgundy
Masters in Computer Vision

Computer vision and Robotics with ROS

- Sandeep Manandhar
- Mohamed Salah

Introduction

- AR-tag detection
- Face Detection
- Face Recognition
- SMACH

Sensors

- Encoders
- IMU
- LiDAR
- Kinect

Sensors

- Encoders
  - Counts pulses in rotation
  - Incremental distance
  - Odometry

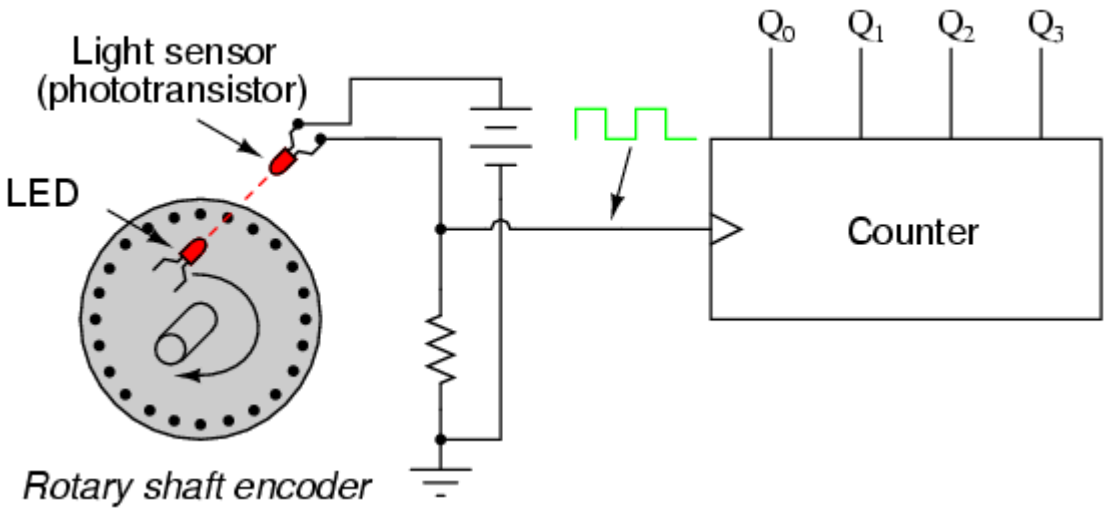

Fig 1: Rotary encoder
Image source: allaboutcircuits.com

Odometry

- Backlash
- Slippage
- Drift
- Worn out wheels
- Travelling surface

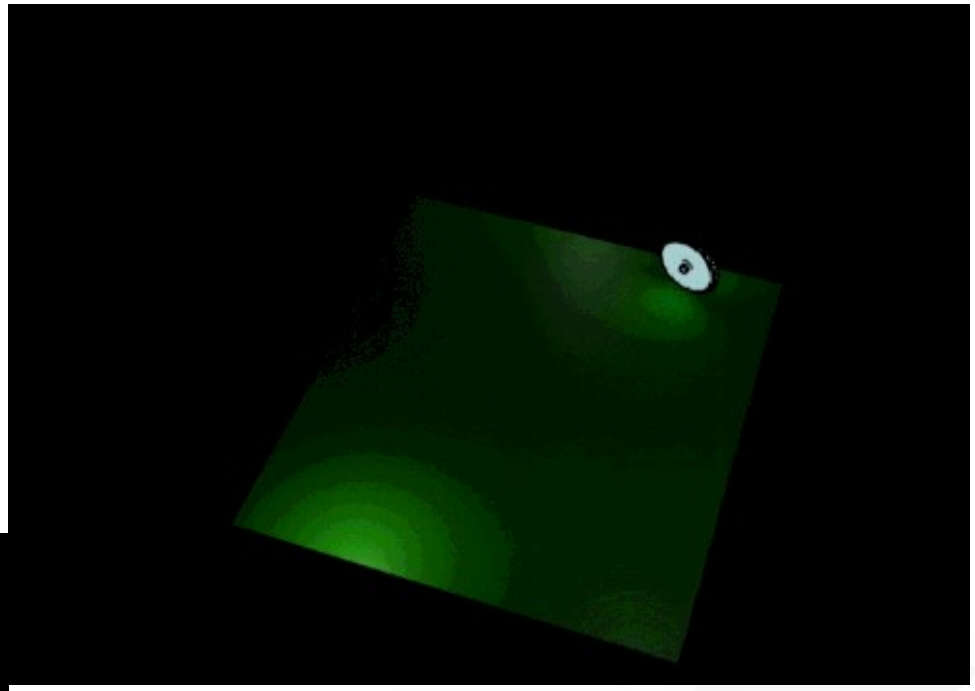

Fig 3: Turning in a curve

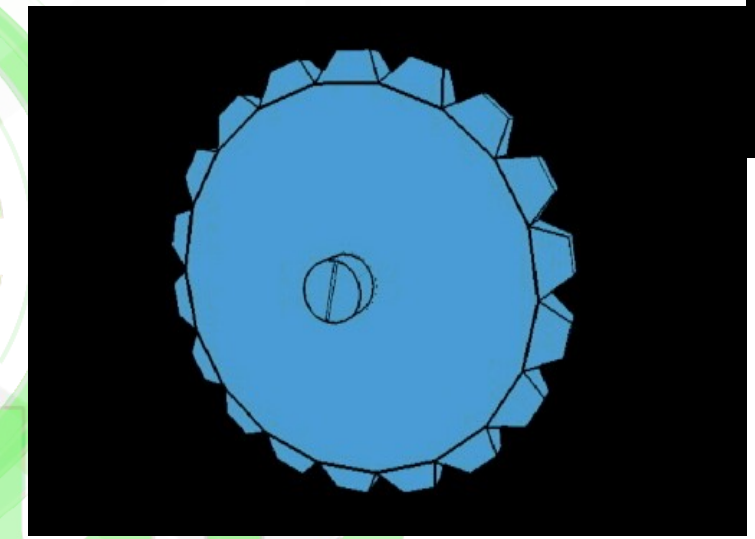

Fig 2: backlash

Sensors

- IMU
  - Inertial Measurement
  - Accelerometer and Gyroscopes
  - Measure acceleration and rotation around axes

Sensor

- LiDAR
  - 360 degrees laser scan
  - 2D point cloud
  - SLAM
  - Distance+heading angle

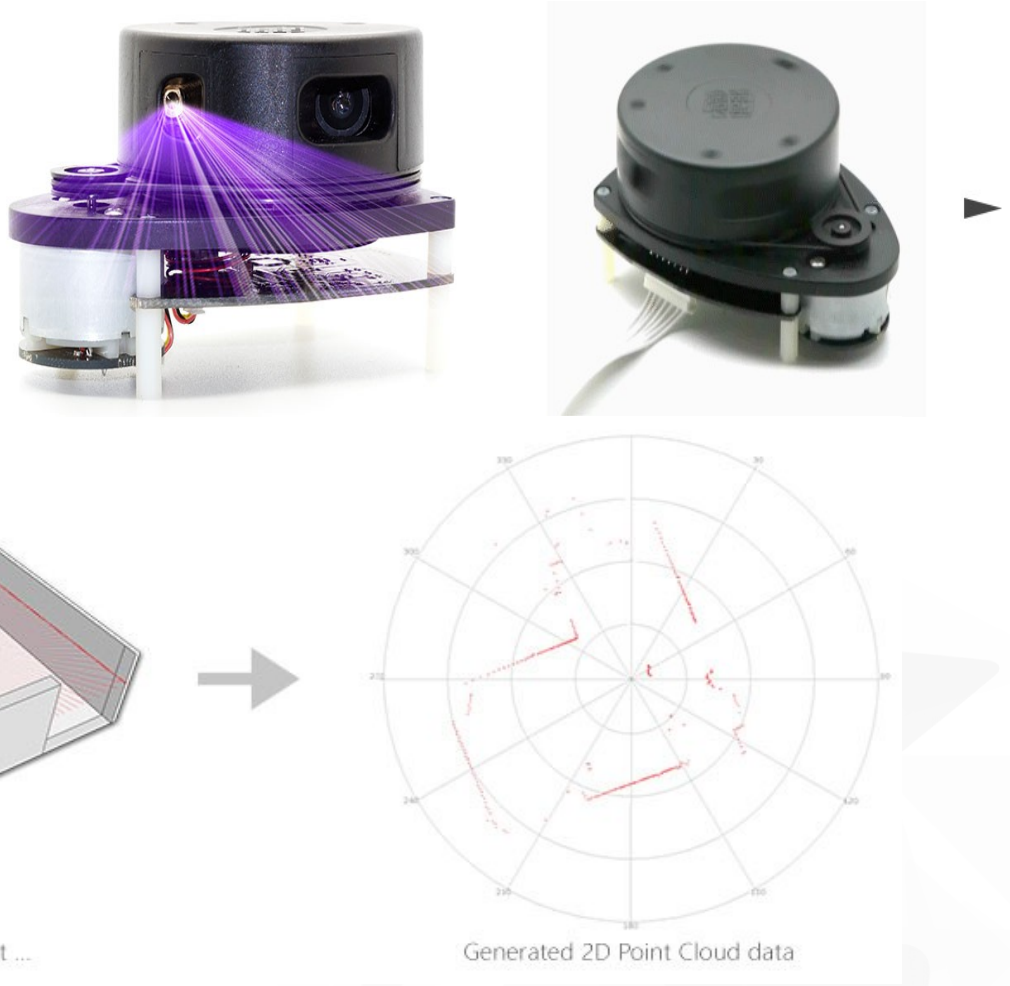

Fig 4: RpLiDAR
Image Source: http://www.slamtec.com/

Sensors

- Kinect
  - RGB-D sensor
  - Structured light
  - Depth map

MAP

Defined as:
-Image: my_map.png
-resolution: 0.1
-origin: [0.0, 0.0, 0.0]
-occupied_thresh: 0.65
-free_thresh: 0.196
-negate: 0

Fig 5: Map view 1

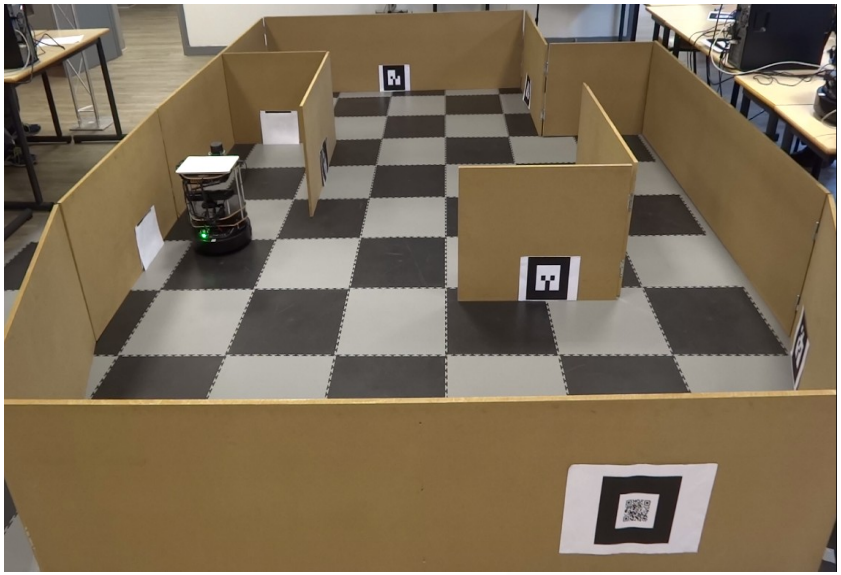

Fig 6: Map view 2

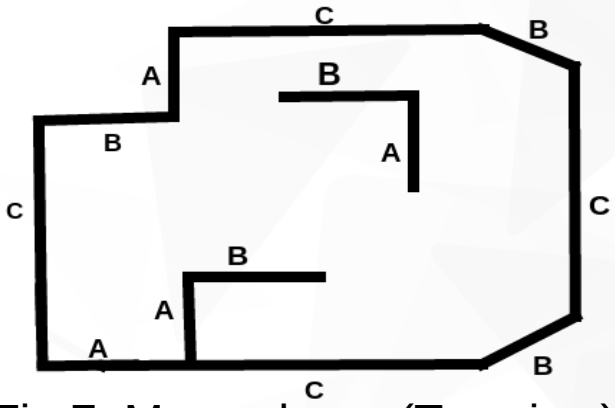

Fig 7: Map scheme(Top view)

TFs

- Transformations
- Describes world with different frames
- Defines Pose
  - Position
  - Orientation

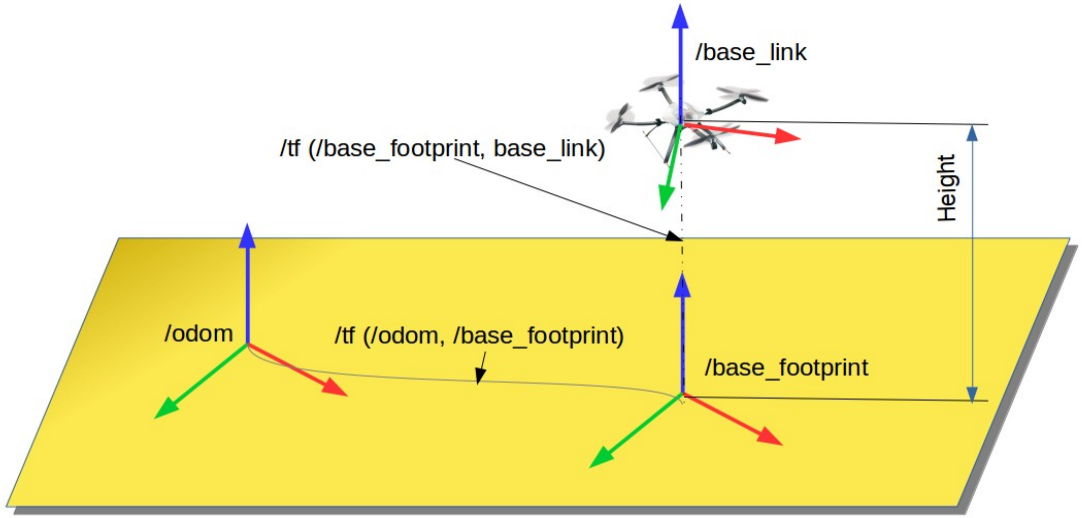

Fig 8: Frames and tf
Image Source: wiki.ros.org

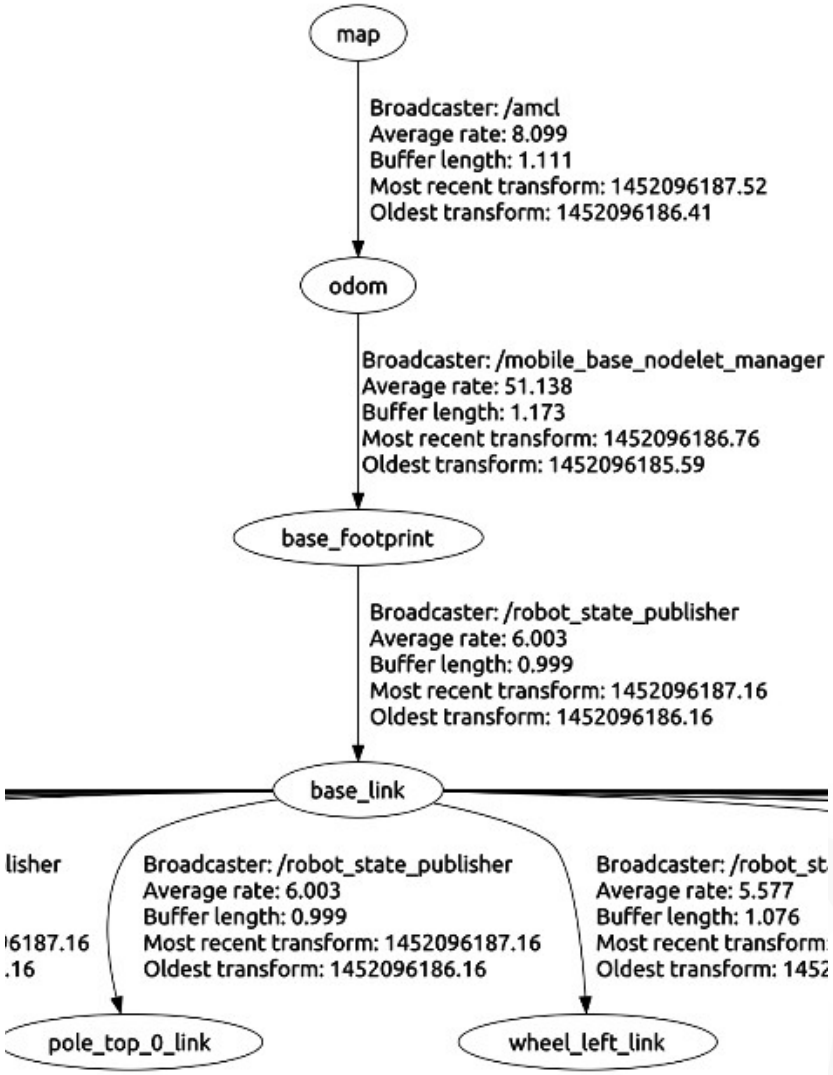

Fig 9: Tf tree

AR-tags

- Bi tonal Pattern
- 7x7 black and white squares
- Known Geometry
  - Position
  - Orientation
  - ID

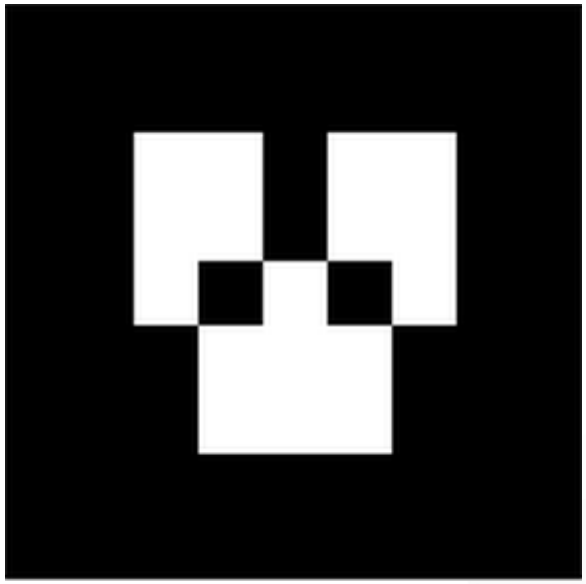

Fig 8: AR tag

AR-tags

- 4 corners for Homography

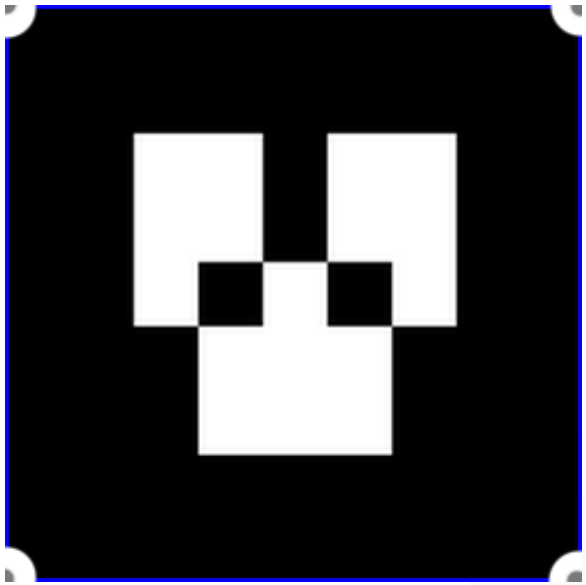

Fig 9: 4 corners

AR-tags

- 2 unit offset

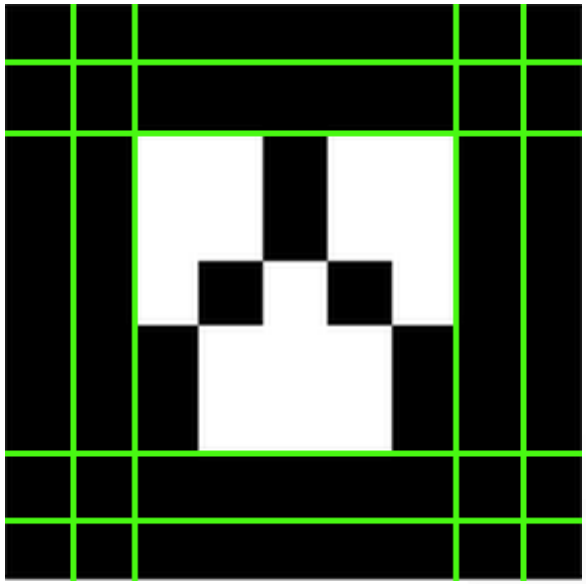

Fig 10:offset to pattern

AR-tags

- 5x5 bitonal pattern

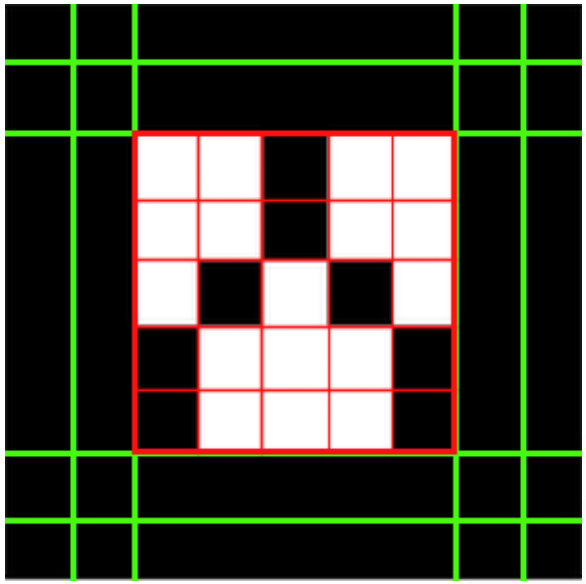

Fig 11: 5x5 pattern
in Red grid

AR-tags

- Detection
  - Library: ar_track_alvar
  - Message: ar_pose_marker

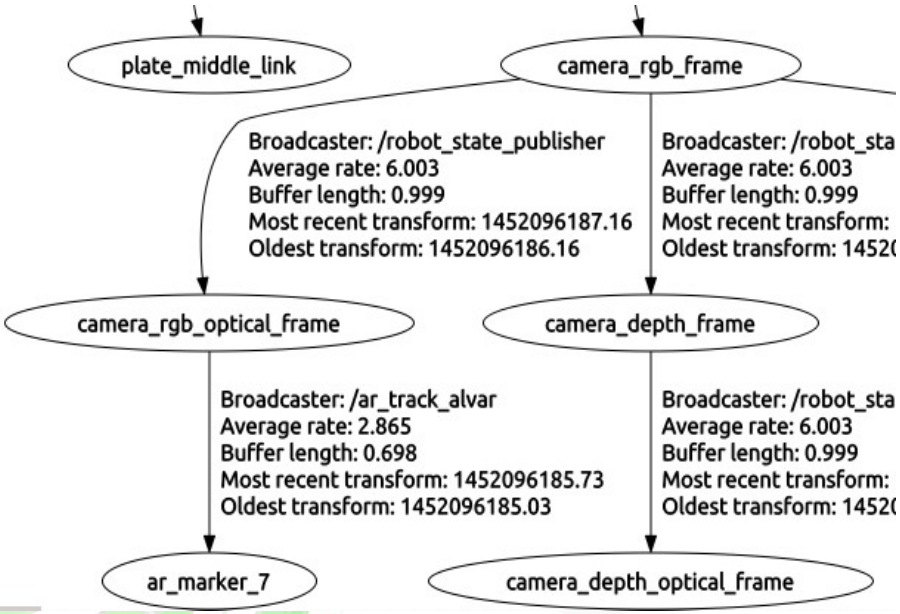

Fig 12: AR tag in tf tree

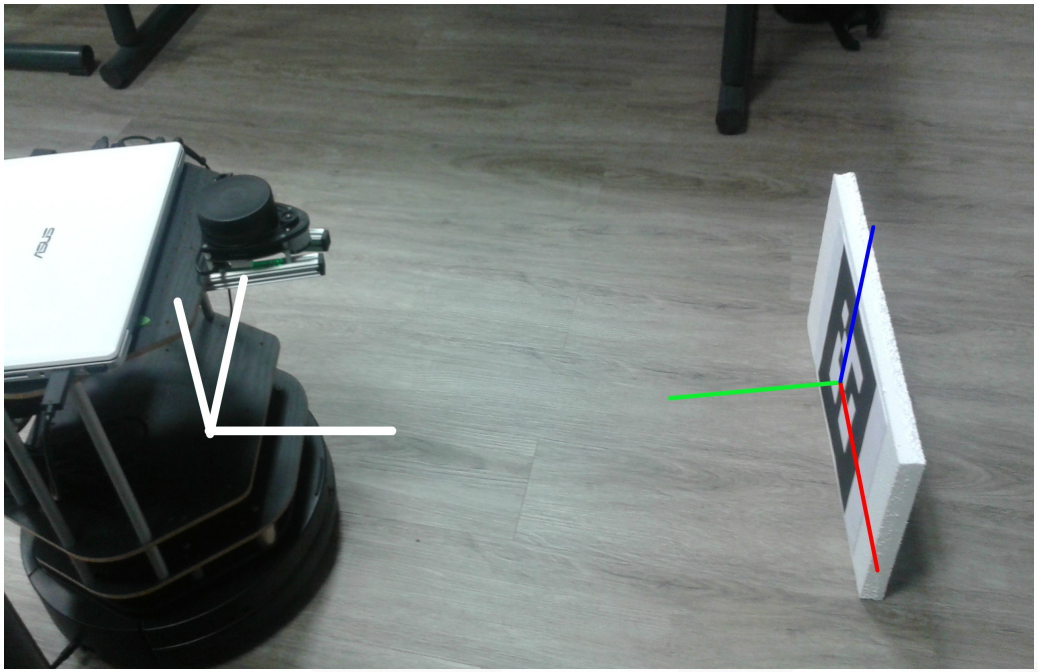

Fig 13: Robot viewing AR tag

Pose Estimation

- Initial State of Robot
  - Robot knows nothing
  - Local map disoriented
  - High covariance

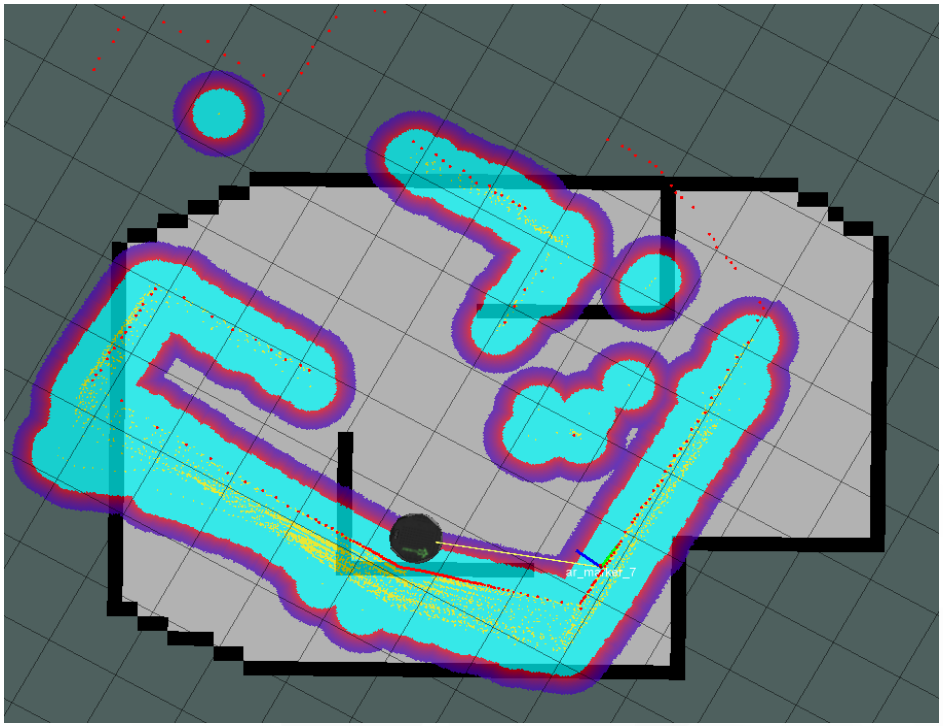

Fig 14: Disoriented Robot

Pose Estimation

- From 3 known tags

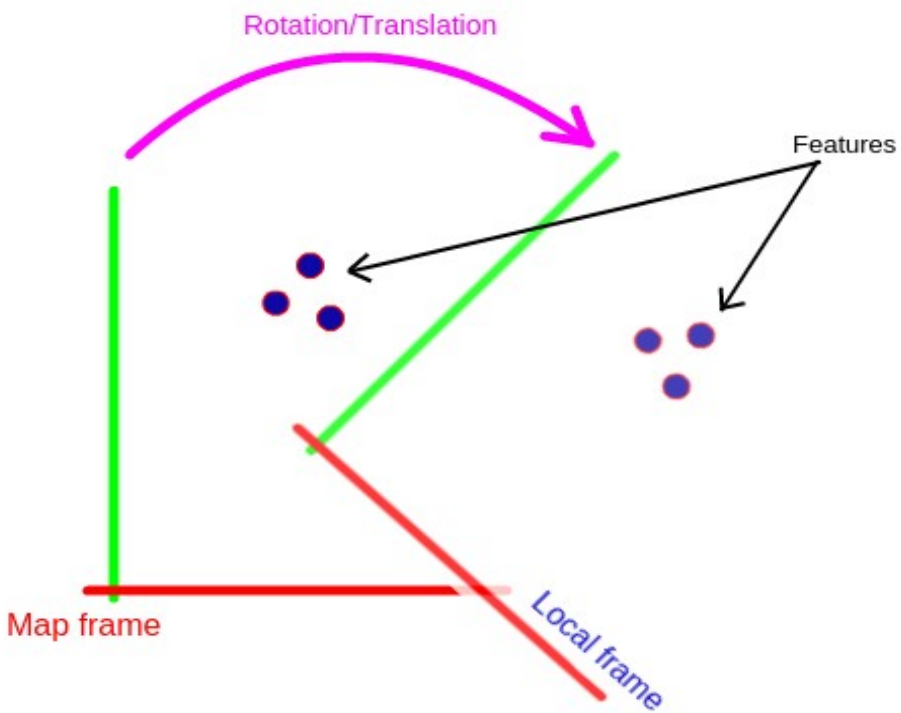

Fig 15: Global and Local frame

Fig 16: viewing 3 tags

C = AB'
USV' = svd(C)

R = inv(UV')
T = -R* T_A + T_B

Pose Estimation

AR-tag

Fig 17.1: AR-tag in a world

Pose Estimation

AR-tag

Turtlebot

Fig 17.2: AR-tag and a robot in a world

Pose Estimation

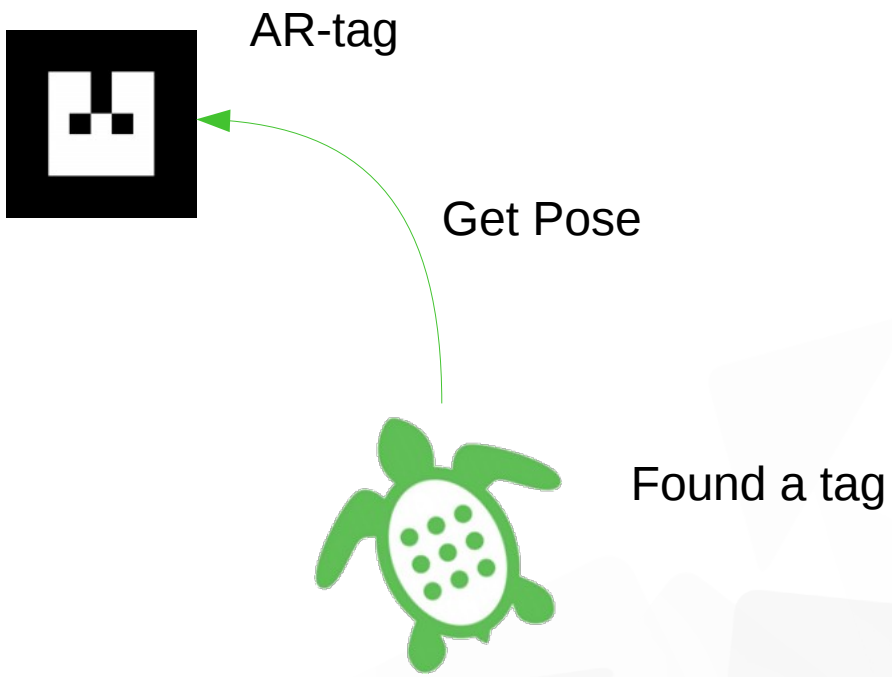

Fig 17.3: Robot meets AR-tag

Pose Estimation

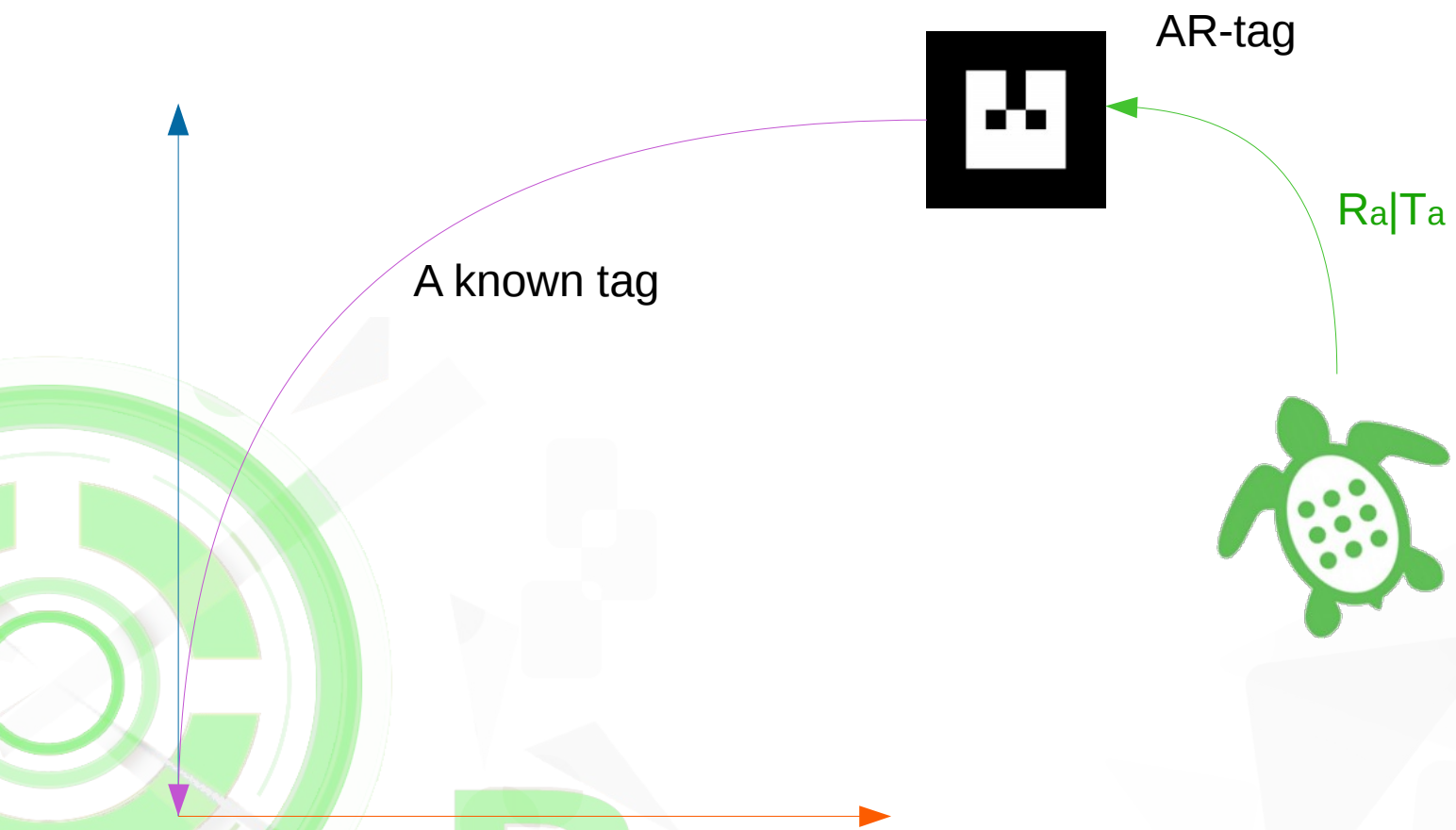

Fig 17.4: AR-tag is known from previous calibration

Pose Estimation

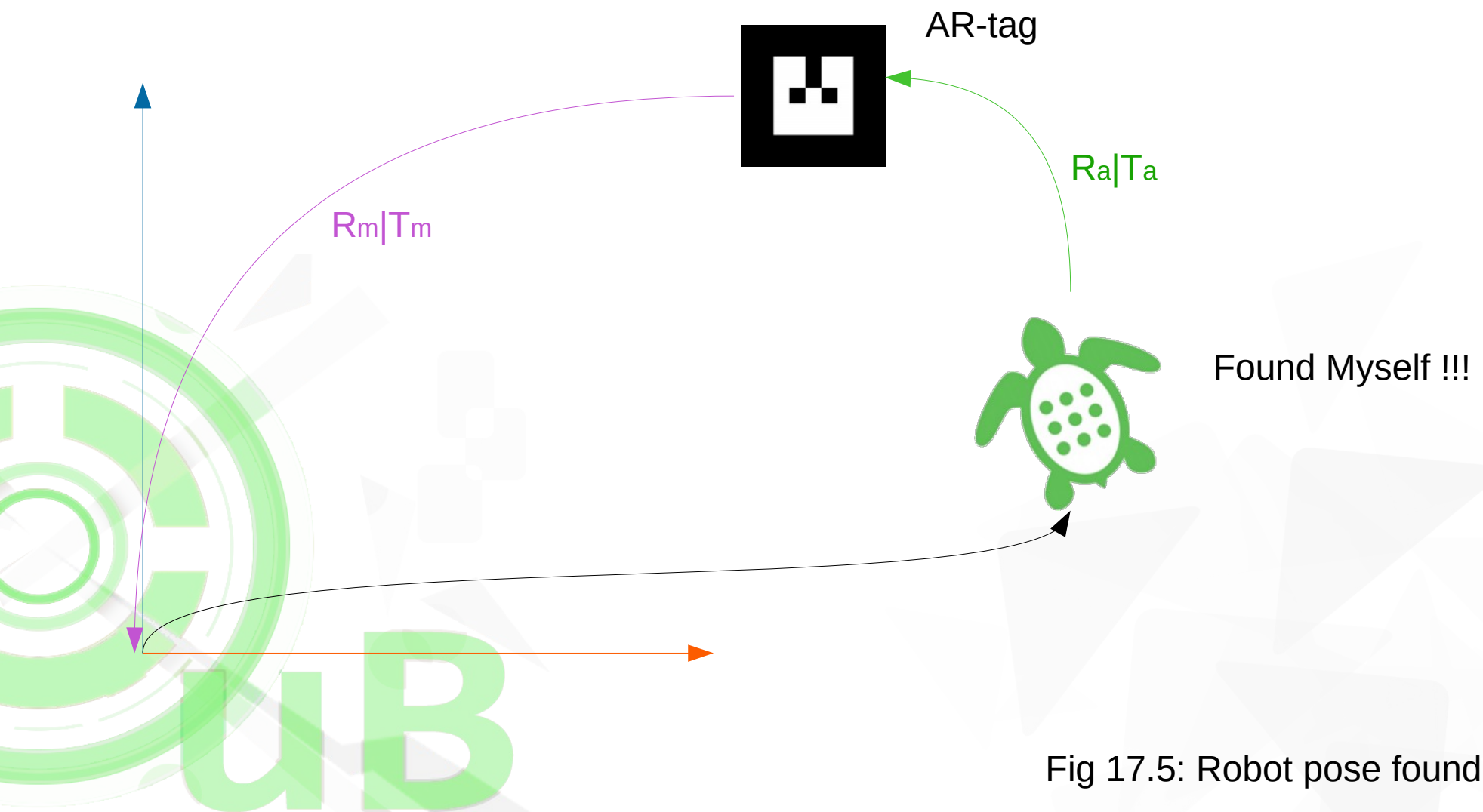

Fig 17.5: Robot pose found

Pose Estimation

- From 1 known tag

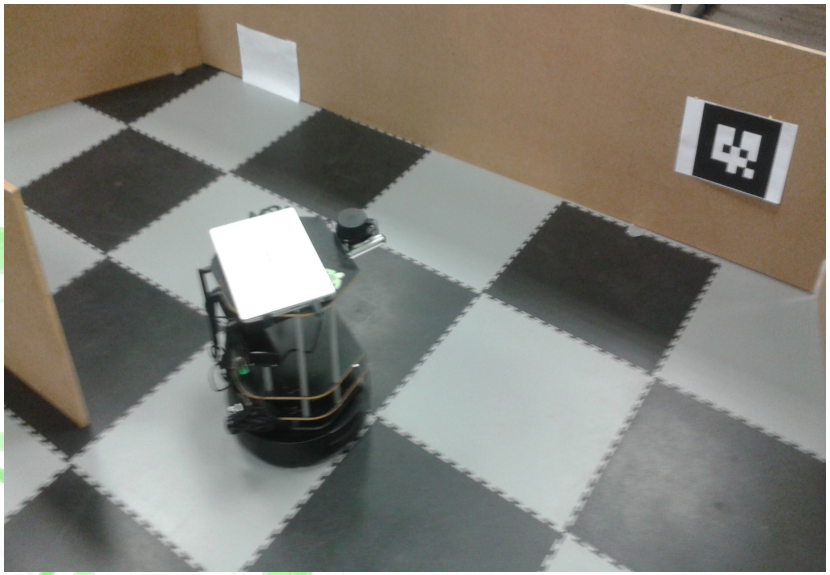

Fig 18: viewing 1 tag

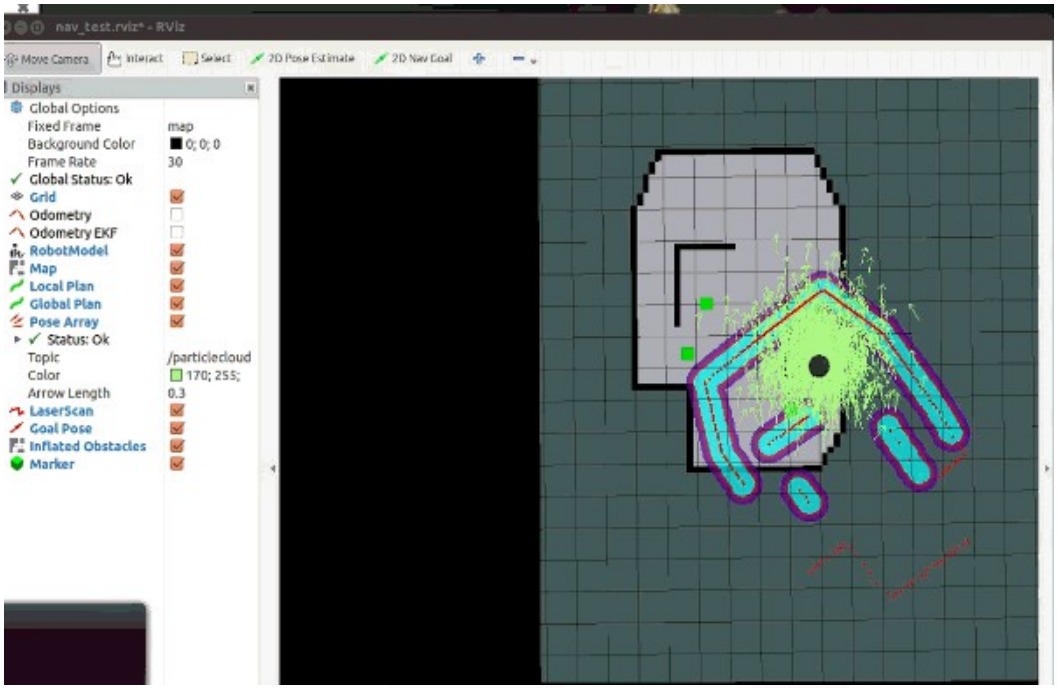

Fig 19: Robot oriented in map

Pose Estimation

- From 1 known tag

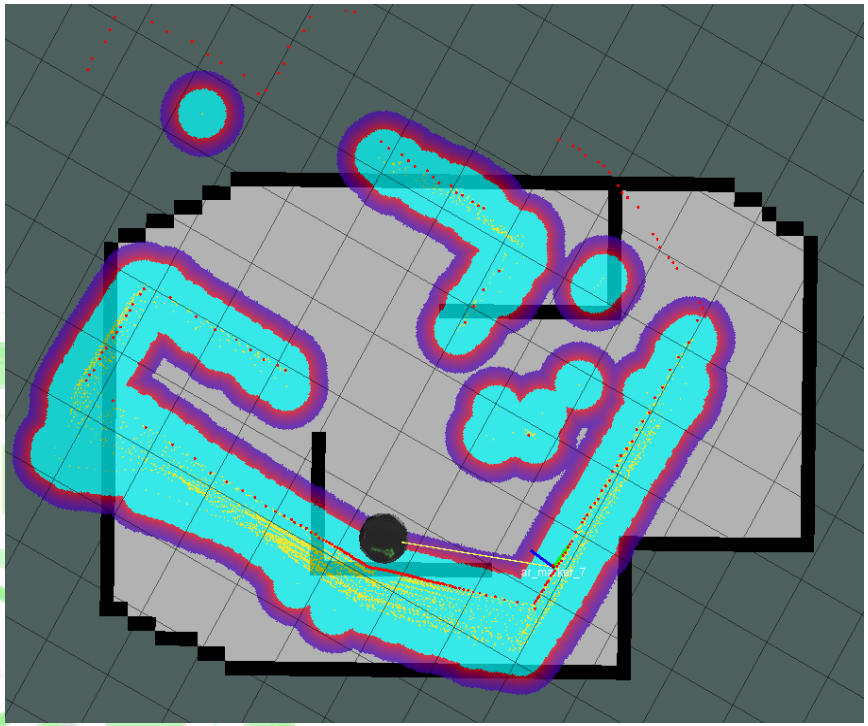

Fig 20.1: Disoriented

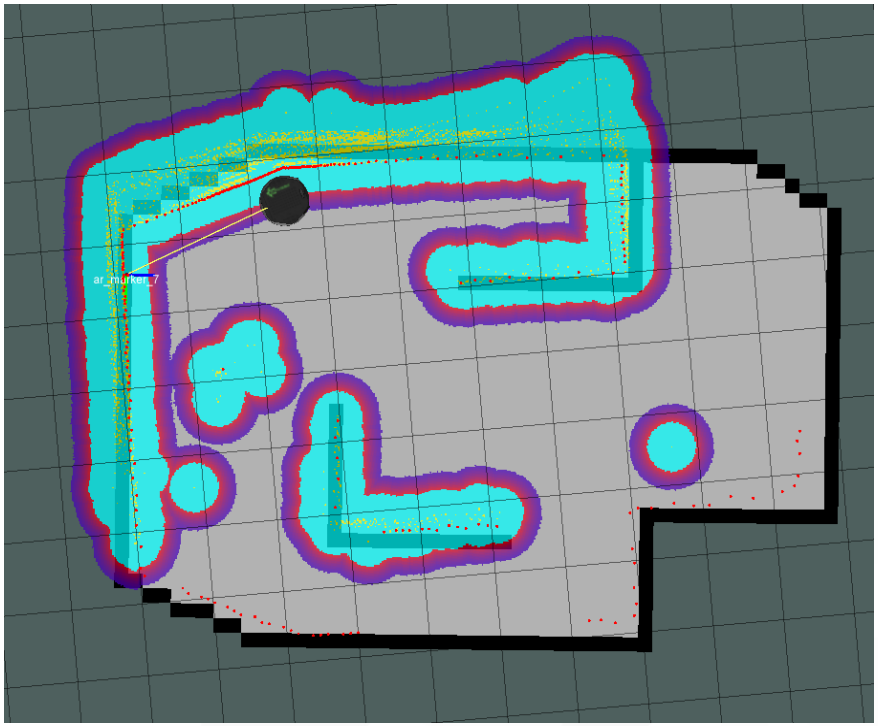

Fig 20.2: Oriented

Tasks: Face Detection

- OpenCV built-in face detector
- Based on Viola-Jones algorithm
- A cascade of classifiers (3) with Haar-like features for target object (Faces)

Tasks: Face Detection

- Which Camera? Kinect
- Package uvc_camera
- provides drivers for USB Video Class (UVC) cameras
- Deprecated!
- CvBridge is used to convert from CvImage to ROS Image Message

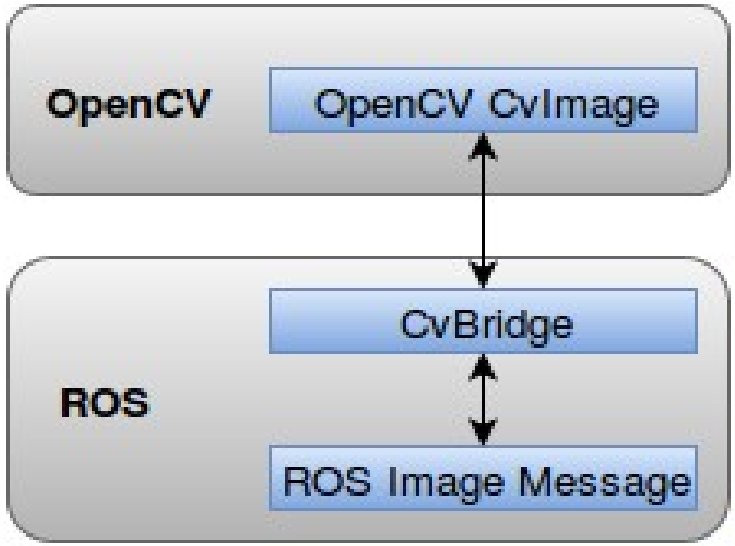

Fig 20: CvBridge

Tasks: Face Detection

- OpenCV image is converted to grayscale
- Histogram Equalization is applied for better range of intensities
- Dispatch ouput image to each classifier

Tasks: Face Detection

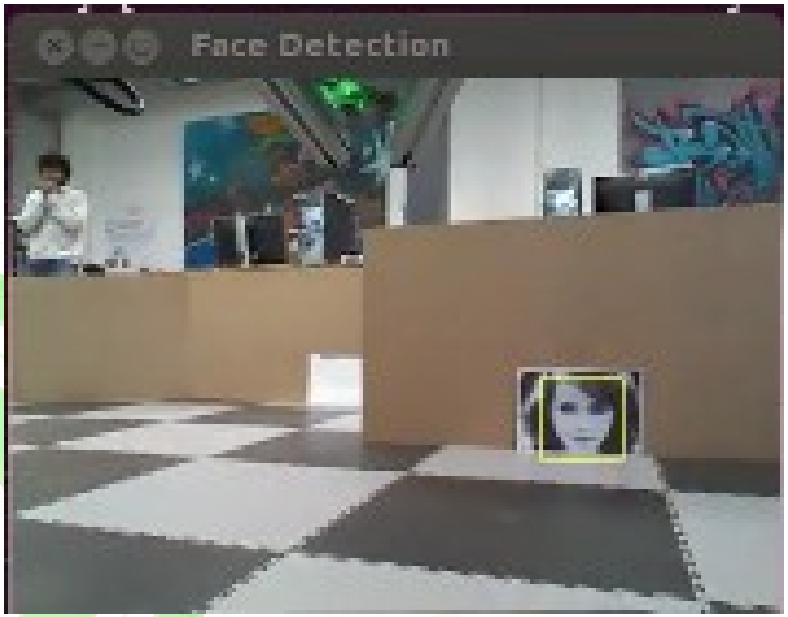

Fig 21.1: Face detected

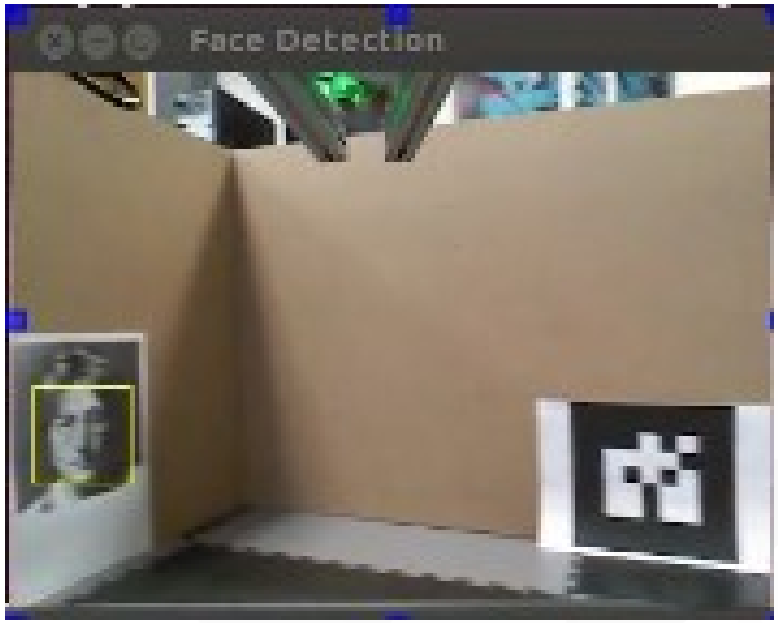

Fig 21.2: Face detected

Tasks: Face Recognition

- face_recognition is written in C++ using actionlib package
- Based on Eigen faces
- Fserver(SimpleActionServer) provides all face recognition functions (train, recognize, shutdown)
- Fclient(SimpleActionClient) provides a functionality to contact with Fserver

Tasks: Face Recognition

- We use Fserver, launch it as a standalone node
- We build our Fclient in Python

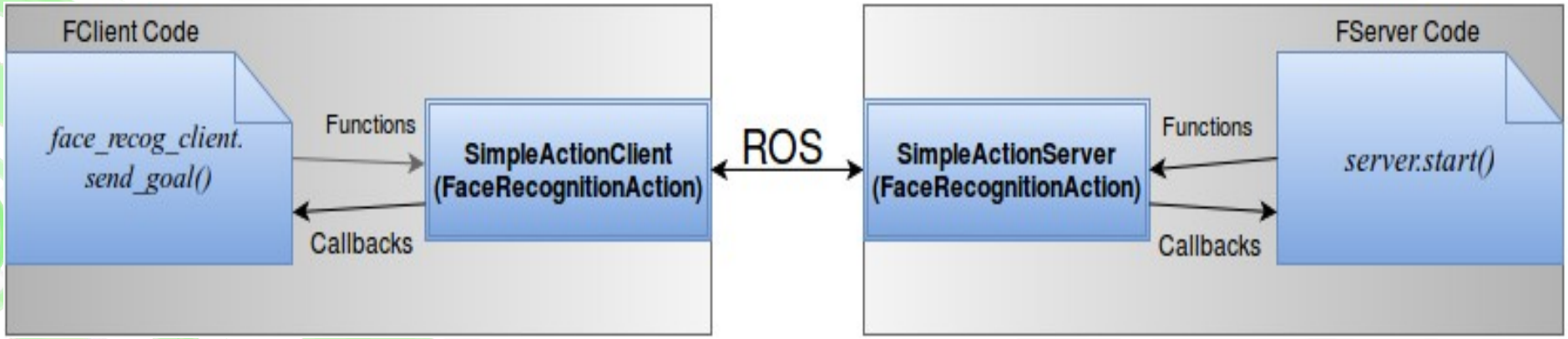

Fig 22: Face server and face client

Tasks: Face Recognition

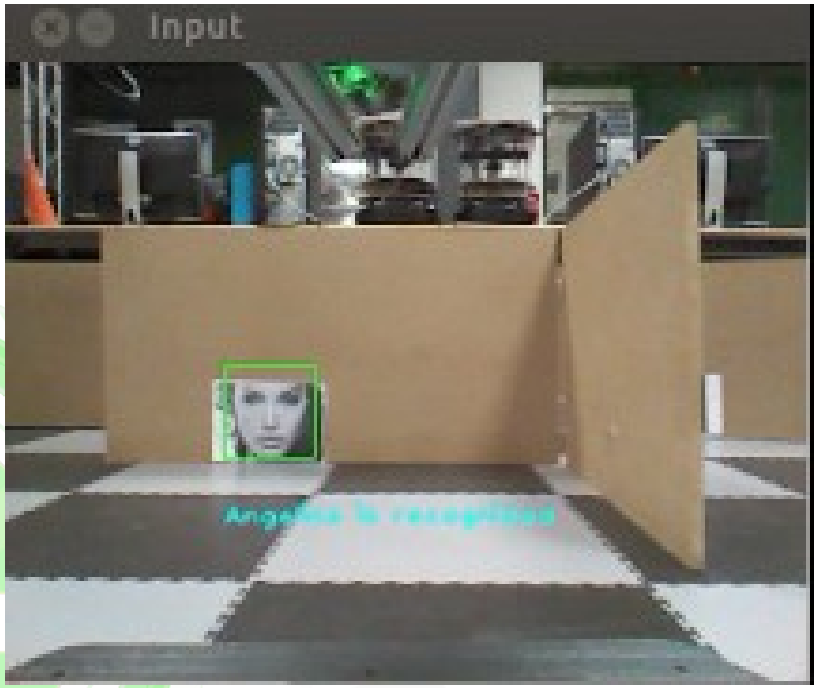

Fig 23.1: Face recognised

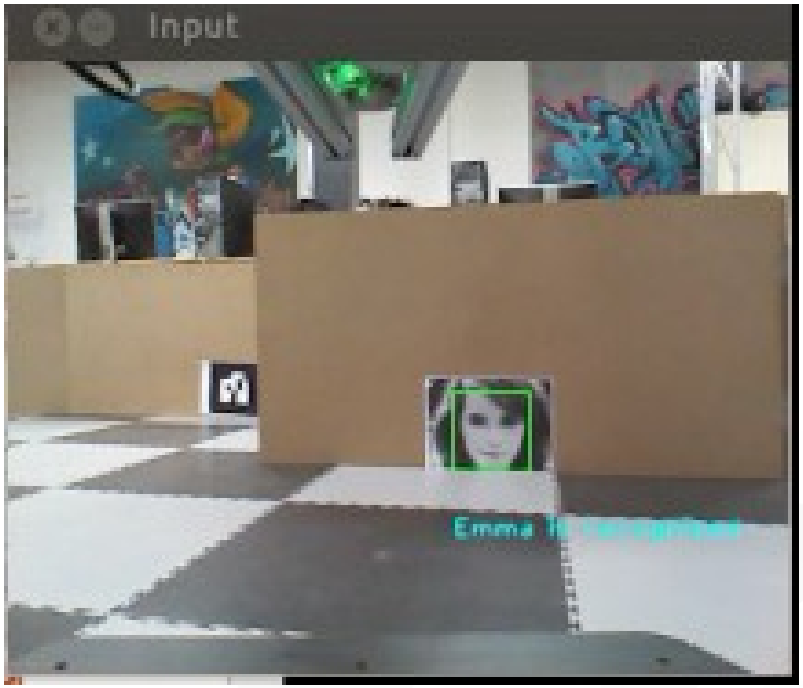

Fig 23.1: Face recognised

Task Execution

- Two commons frameworks (SMACH, pi_trees)
- SMACH follows finite state machine model of computation
- pi_trees adapts behavior trees paradigm
- Both provide minimum set of key features:
  - Task priorities
  - Pause and Resume
  - Task Hierarchy
  - Condition Manipulation
  - Concurrency

SMACH

- Pure Python library
- Each task can be represented as a State with transition and outputs
- Each State is an object instantiation of smach.State that must override execute() method
- Each State should return an outcome that define its next transition
- Each State can have an input and output

SMACH

- SMACH provides a number of defined State containers for different application
  - Concurrence container
  - Sequence container
  - Iterator container

SMACH

- SMACH provides a number of defined State subclasses for different application
  - SimpleActionState
  - MonitorState
  - ServiceState
  - ConditionState

SMACH: Pose Estimate

- A smach.State child class
- Wrap Pose Estimation Behavior
- Stay in the state till it sees AR-Tag 7
- Start concurrent functionality once pose is estimated

SMACH: Concurrent State Machine

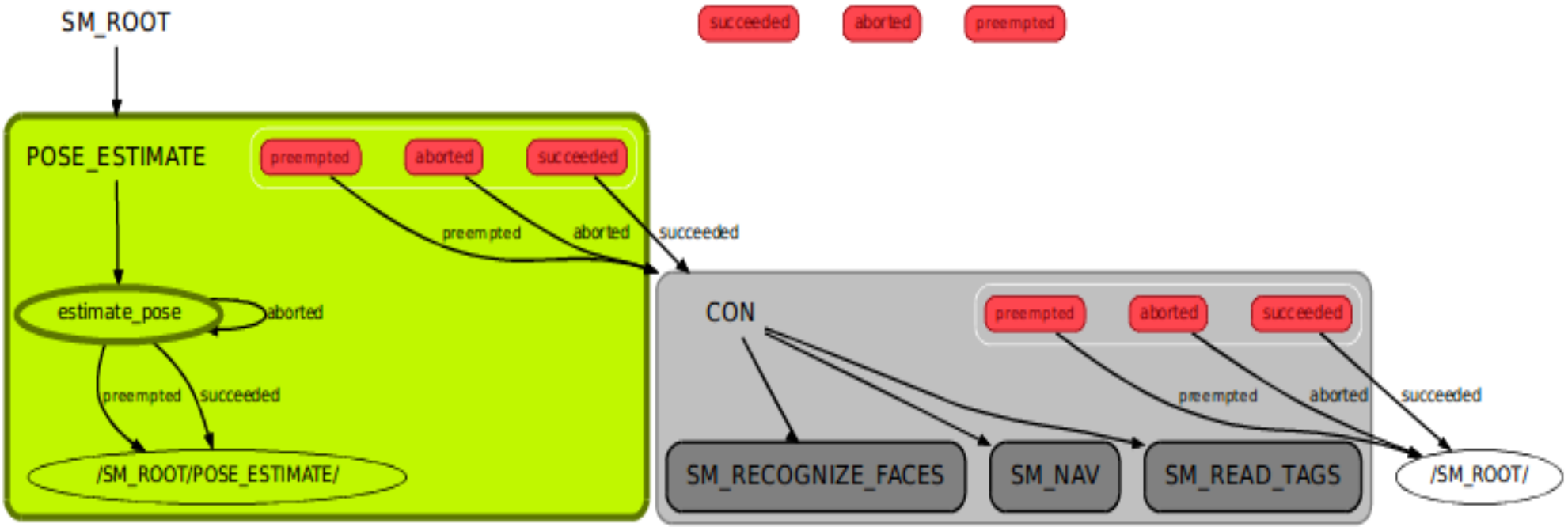

Fig 24: State Machine

SMACH: Navigation State

- smach_ros.SimpleActionState? DynamicGoal
- smach.State child class
- Userdata: shared list of poses
  - Predefined primary targets
  - Secondary targets appended by AR-Tag
- Aborts when list is empty (All targets visited)
- Preempt all others running states

SMACH: AR-Tag State

- smach_ros.MonitorState? Input? Output?
- smach_ros.MonitorState child class
- Subscribe to topic /ar_pose_marker
- Message Type: AlvarMarkers
- Userdata: shared list of poses
  - Append detected tags' poses
- Preempted when navigation is finished

SMACH: Face Detection State

- smach_ros.MonitorState?
- smach_ros.MonitorState child class
- Subscribe to topic /camera/rgb/image_color
- Message Type: Image
- Preempted when navigation is finished

SMACH: Face Recognition State

- Python subprocess? “I am SMACH, use me!”
- SimpleActionClient interface
- smach_ros.SimpleActionState?
- smach.state child class
  - More modularity, every task is in its own class file
  - Understand How SMACH works?

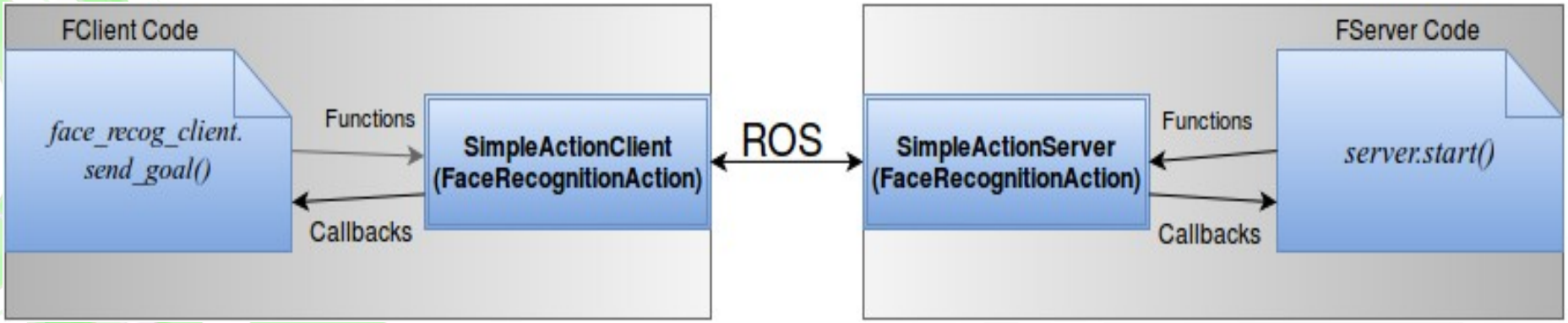

Fig 25: Face server and face client

RViz Utilities

- ROS main visualization utility
- Add visual markers for primary at initial start time
- Used during run time to add visual markers for secondary targets

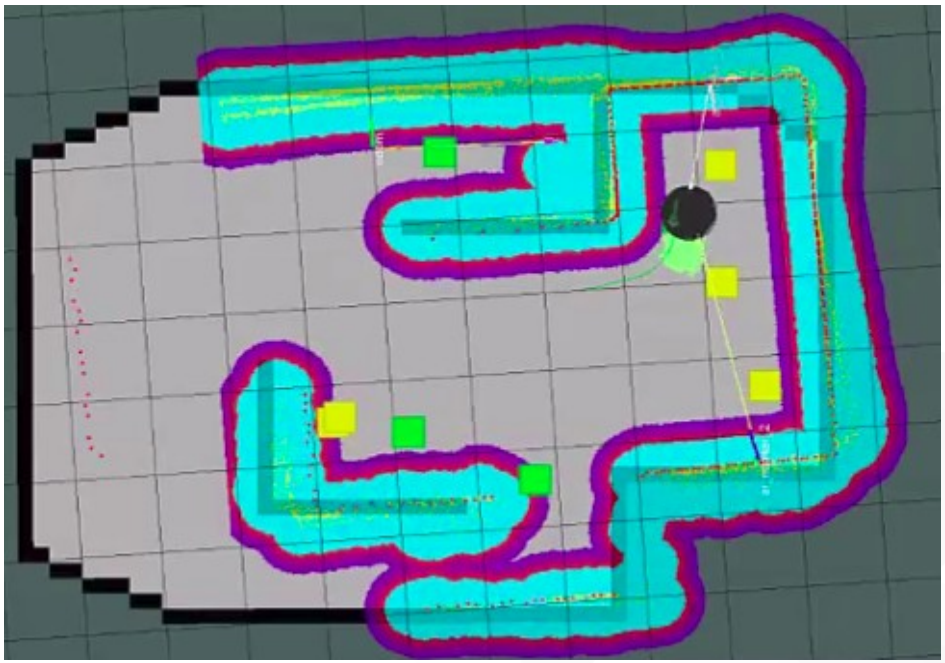

Fig 26: Rviz Map

RViz Utilities

- RvizUtils class
- Singleton Design Pattern

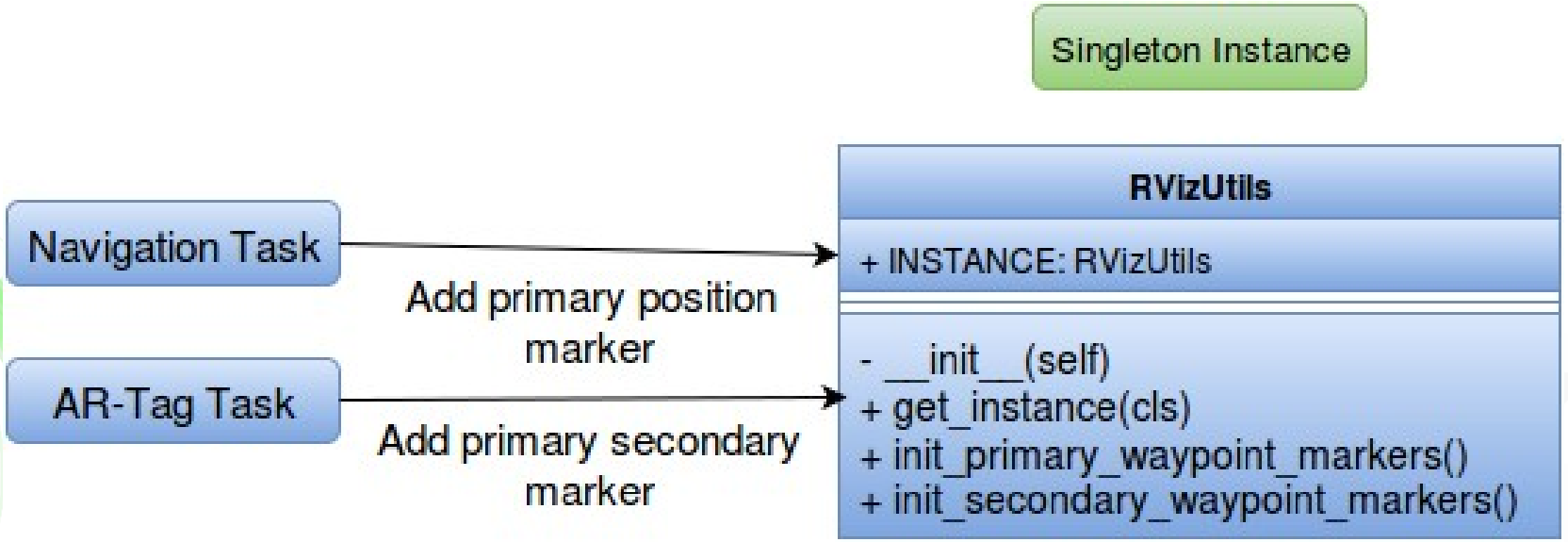

Fig 26: Rvizutils class

Sound Utilities

- Receive Feedback from our robot AR-Tag and Face Recognition
- Package sound_play
- Provide a ROS node that converts ROS message into sound
- Supports 3 types of sound message :
  - Build-in sound
  - OGG/WAV sound file
  - Speech synthesis using festival (Text to Speech system)

Sound Utilities

- Package sound_play drawbacks:
  - Week and wrong documentation
  - You cannot change the voice easily, change it for the whole system
  - Weird behavior if used by multiple clients
  - No way to tell when sound stops playing
  - Bad message queue implementation

Sound Utilities

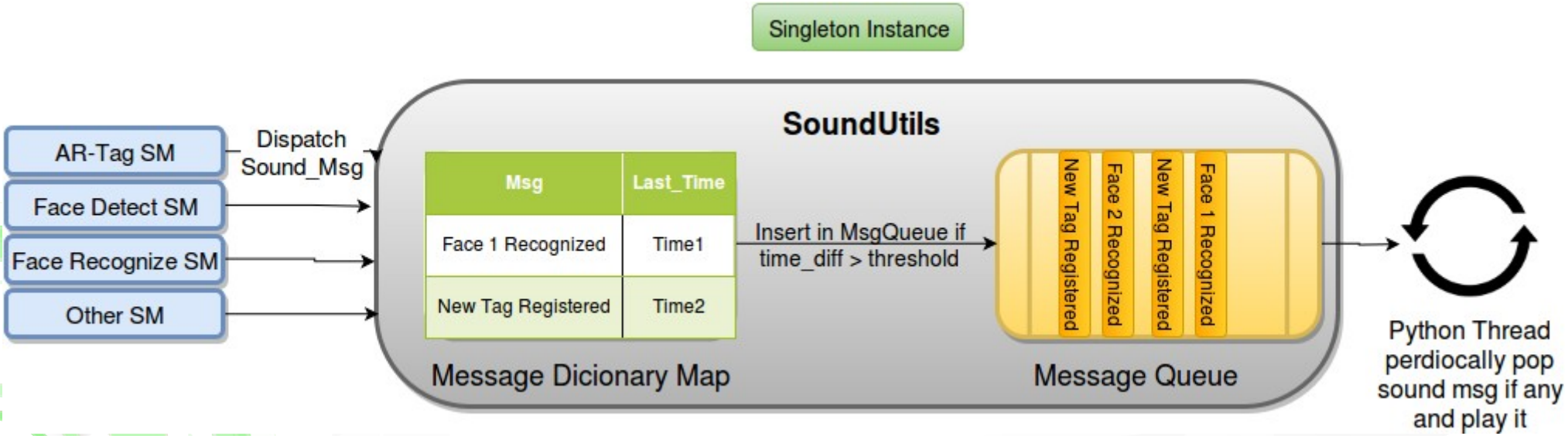

Fig 27: Sound utils

Thank you …

NOW DEMO !