Obstacle Detection and Avoidance

Miguel A. Duenas

November 7, 2024

Release 2.0

Description

The purpose of this document is to explain how my obstacle detection and avoidance algorithm works and how it navigates the Autonomous Ground AGV (AGV).
The idea is simple:

- If a static object is detected, rotate until its cleared. Once cleared continue moving to goal.
- If a dynamic object is detected, stop until the object moves out of the AGV view.

The algorithm depends on two ROS topics types: Lidar and Odometry. For each topic, we subscribe a callback function, cbLidar and cbOdometry.

The following slides will explain what each function does and a diagram showing the data flow.

cbLidar

cbLidar is responsible for detecting obstacles and moving the AGV until the object is not detected.
Using the lidar data, it reads the ranges. When it detects a short range in the direction of the front of the AGV, it takes the following actions:

- Sets a global flag that an obstacle is detected.
- Stop the AGV.
- Rotate the AGV.

Once it stops reading short ranges from the lasers in front of the AGV, it takes the following actions:

- Move the AGV a short distance in the current direction.
- Set global flag that an obstacle is NOT detected.

These actions are managed by a state machine. The state machine has the following states: INIT, STATIC, DYNAMIC, and SCANNING.
Here is a description on what each state does:

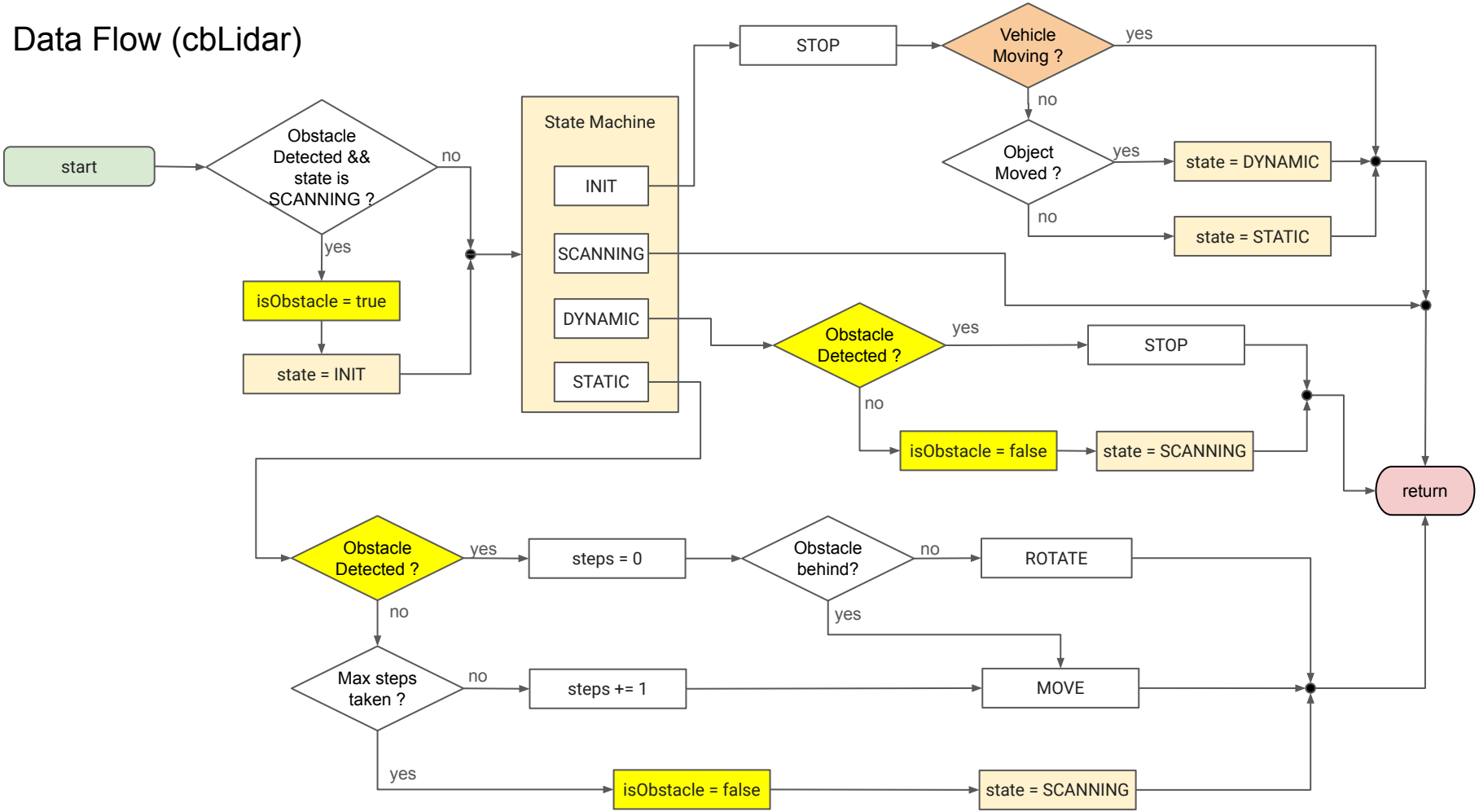
- INIT
  Determine if obstacle is static or dynamic (moving).

- STATIC
  Rotate and Move AGV until the obstacle is no longer obstructing our path.

- DYNAMIC
  Stop AGV until obstacle is not longer obstructing our path.

- SCANNING
  Read scan data and look for obstacles.

Data Flow (cbLidar)



cbOdometry

cbOdometry is responsible for calculating the heading to our goal and move the AGV in that direction. It uses the pose quaternion to calculate the euler angles and get the yaw of our AGV. It also uses the pose position and the goal position to calculate the “angle-to-goal”. Using this angle-to-goal and the yaw value, we can calculate if we are heading in the correct direction.

Once we have these values, it takes the following actions: Stop, Spin, or Move.

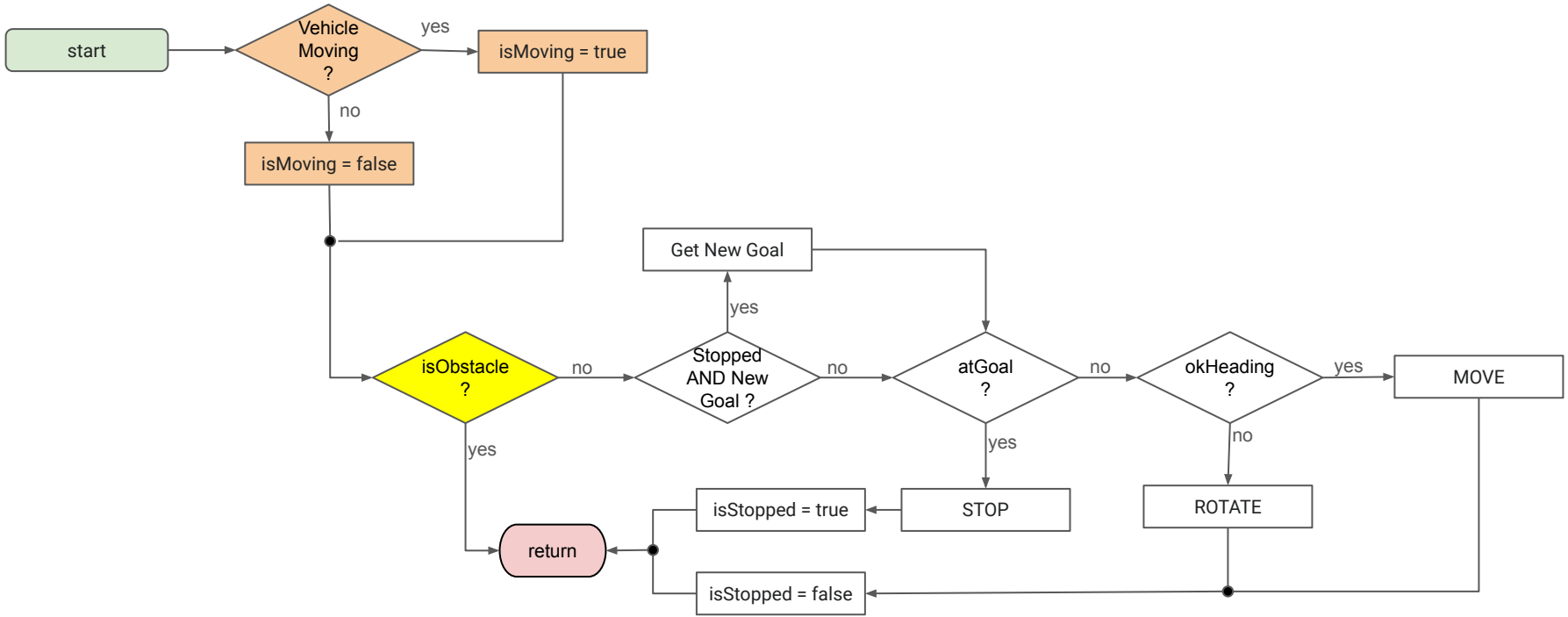
- Stop
  Current position and goal are the approximately same location.

- Spin
  The angle difference between angle-to-goal and yaw is greater than 0.1 gradians.

- Move
  Default action if we are not stopping nor spinning. In other words, we are heading in the correction direction, so move.

Note: If an obstacle is detected by the cbLidar, the cbOdometry function will return immediately after it gets called. We don’t want both callbacks controlling the AGV at the same time.

Data Flow (cbOdometry)



Errors / Assumptions

Odometry Error

This algorithm is only using odometry to calculate position and direction. Overtime, odometry builds error and this will affect the navigation of the AGV. This error can come from

- Incorrect wheel geometry settings (radius, pose)
- Wheel slippage, like when the vehicle rotates in place.

A potential solution to correct these errors is by using an IMU. IMU gives us the correct orientation of the AGV, so this can correct the heading. Using the accelerometers, we can detect if we are moving. And using the gyroscopes, we can detect rotations. So using these information we can detect wheel slippage or rotations.