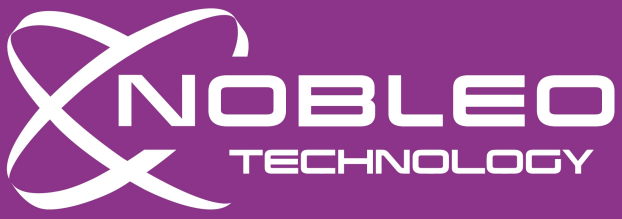
Mobile Robot Navigation in ROS2

Motion Controllers Comparison

César López
Mukunda Bharatheesha
Bram Odrosslij
Frank Sperling

December 23, 2022



Contents

1 The Mobile Robot Navigation Problem

Mobile Robot Navigation Problem

In a fast-changing world, automating human work is becoming a necessity to keep businesses running. Not only does automation prevent humans from doing dangerous and dull work, it also provides a solution to the labor shortages seen in this line of work. Among other technologies, we develop robust autonomous systems, specifically Autonomous Mobile Robots (AMRs), for reliable operation in industrial applications like surveillance, inspection, logistics, and agriculture. In this article we want to highlight one of the major technical challenges we face in developing this mobile robot navigation technology. This challenge deals with deciding the steering and velocity commands an AMR needs to perform a given task, whose solution is non-unique and far from trivial.

Autonomous Guided Vehicles (AGVs), which are the predecessors of AMRs, were developed in the 1950's [31]. These vehicles relied on fixed infrastructure being present in the environment, to be used as guidance, hence the name. On one hand, the algorithms required inside the vehicle were simple. However, installation costs were high, and therefore AGV usage was limited. In addition, due to the simplicity of the solution, it could be applied only to use cases where high accuracy was not required and to environments with little to no uncertainty.

Next to the growing demand for robotized solutions, additional challenges emerged that AGVs could not cope with. For instance, factories, where robots and people share the same space, would require the AGVs to detect obstacles in their surroundings to avoid collisions. In the presence of static obstacles that were not initially present, it would also require that the vehicle could find a way to circumvent these obstacles. In turn, this also would mean that the robot needs to localize itself in the environment at any location, and not only at the

places where the fixed infrastructure allows it.

To tackle those challenges, AGV technology evolved into AMR technology [3]. This also means that the complexity of the algorithms needed in the vehicle increased significantly. One of the main enablers of AMRs has been the advent of the open-source Robot Operating System (ROS) [14], and together with the declining costs of electronics, the total cost of ownership for an autonomous mobile platform decreased dramatically, making it more feasible to use the technology in a wide variety of applications. ROS in its core is a middleware framework, which allows different software components to communicate not only within a single robot but also across multiple machines. On top of that, the open-source community has developed a stack of software modules offering a wide range of functionalities for robot localization, navigation, and data visualization, among others. The community is very active and in recent years a new version of ROS, ROS2 [23], was released, which improves not only the communication layer between modules but also extends the capabilities of its functional modules.

In this article, we will focus on mobile robot navigation. Consider the situation in Figure 1.1, in which a mobile robot needs to reach a predefined area. Along the way to get there, the robot will not only encounter the fixed structure of the environment (i.e. walls, doors, etc.) but also objects which can be moved from time to time, and people, who are constantly on the move. Motion planning algorithms deal with the problem of finding a sequence of velocity and steering commands that will result in the mobile robot successfully reaching the desired target while keeping the integrity of the environment, its actors, and of the robot itself. Successful completion of the navigation task is also subject to certain constraints and performance criteria, which depend on the application. In logistics applications for instance, maximum time, and velocity, are relevant. In other applications like precision agriculture, navigation accuracy plays a major role.

In the coming sections, we will dive into the architecture of the mobile robot navigation software available for ROS2, and in particular into the different motion controllers that are available.

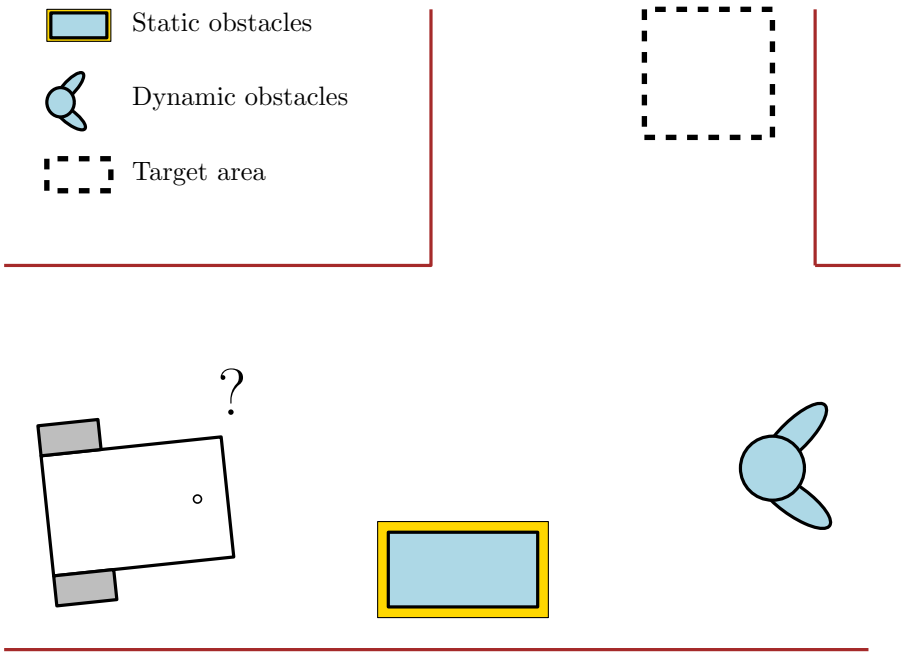


Figure 1.1: The mobile robot navigation problem: How to generate a set of velocity commands to let the robot reach its target area?

2 ROS2 Navigation Architecture

Industry Standard architecture

ROS2 uses the industry standard architecture for mobile robot navigation, which consists mainly of three blocks: environment representation, global path planning, and local motion control. The environment is represented using costmaps, indicating the presence of fixed infrastructure, together with static and dynamic obstacles. Path planning aims to find a collision-free path towards the desired target, and motion control generates velocity and steering commands to guide the vehicle along the planned path towards the target. In this section, we will provide details on how these modules work and interact with each other.

The mobile robot navigation architecture of ROS2 [17] is summarized in the diagram shown in Figure 2.1. The task coordination module is in charge of the high-level coordination of the navigation modules, which in ROS2 is implemented using Behavioral Trees [18]. The global and local costmaps are used to represent the fixed environment, as well as the static and dynamic obstacles [19]. Path planning [21, 30] is in charge of finding collision-free paths between given start and target poses. The motion controller [20, 9] module uses the generated path to create velocity and steering commands that guide the robot toward the desired target. Finally, the odometry control module makes sure that the velocity and steering commands are properly executed in the machine. We will dive into the most relevant modules for the core navigation functionality: costmaps, path planning, and motion control.

For navigation purposes, the environment around the robot is represented by means of cells on the costmap's grid. Cells in such a grid contain a single number that shows whether they are free or occupied. These cells are also "inflated" depending on the robot's dimensions such that the cells around the occupied

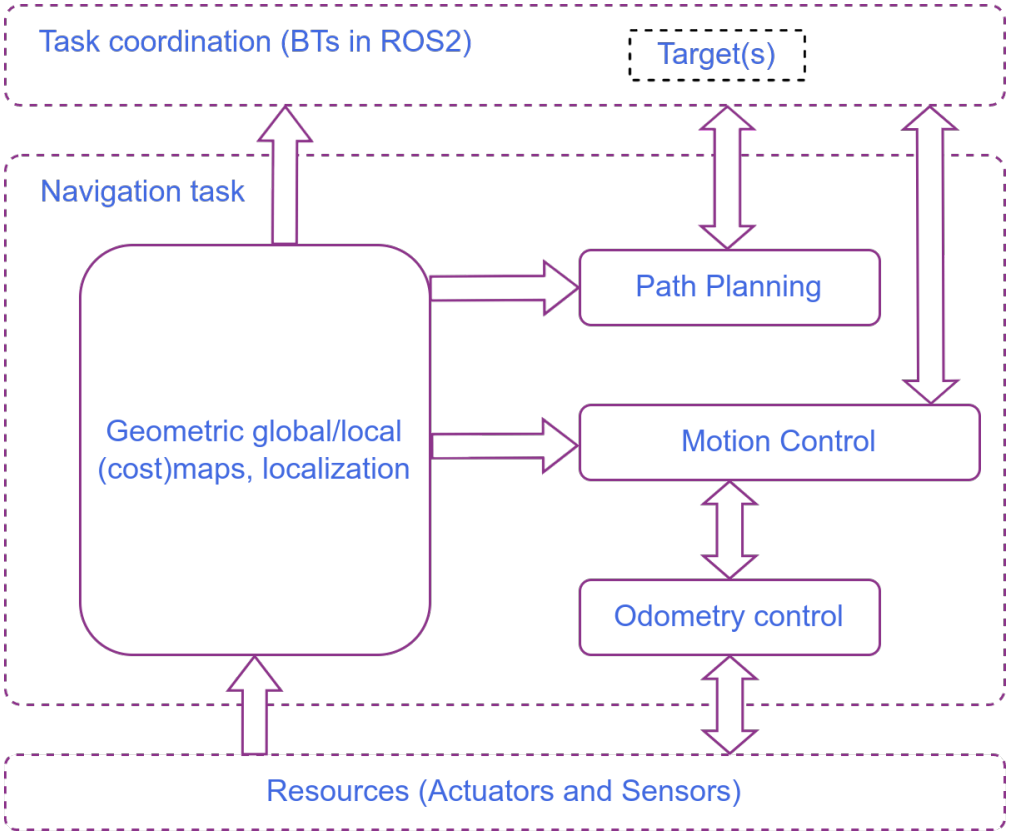


Figure 2.1: ROS2 robot navigation architecture: Environment is described as geometric grid-costmaps of the environment

cells indicate proximity to an obstacle. Exact collisions can not be detected at this stage since they also depend on the robot's orientation, especially for non-symmetric robot footprints. The fixed infrastructure is typically represented in the global costmap, which uses the map of the environment as input. The static and dynamic obstacles are represented in the local costmap, which uses inputs from sensors like LiDAR range finders, sonar sensors, 3D cameras, etc.

The navigation algorithm itself consists mainly of two steps, global path planning, and local motion control. In the first step, global path planning (from now on simply referred to as path planning), the idea is to find a collision-free sequence of poses (i.e. positions and orientations) that connect a starting pose to a target pose. The starting pose is the current pose of the robot with respect to the map's origin (i.e. its localization pose) and the target pose is chosen within the target area. The path planner uses the global costmap together with the geometric footprint of the robot, especially for non-symmetric footprints, to assess

whether it will be in collision with the environment. Path planners initially only use information from the fixed environment, therefore as the robot navigates towards the target pose, it might encounter obstacles that were previously unknown as shown in Figure 2.2. These unknown obstacles can be dealt with by the local motion controller. In addition, path planners take into account the robot's kinematic constraints, i.e. when robots cannot move in all directions and first have to rotate. In fact, the most common driving mechanism in industrial robots is the differential drive, which cannot drive the robot sideways, thus exhibiting a kinematic constraint. Other examples of robots with multiple kinematic constraints are robots steered by their front wheel, which in many cases is classified as an "Ackermann steering" [33]. These robots have kinematic constraints at the rear caster wheels and at the steering wheel.

In the second step, local motion control (from now on simply referred to as motion control) is in charge of generating the actual velocity commands that are executed by the robot. The motion controller uses the generated global path to guide the robot towards the target pose. The resulting sequence of poses, each associated with a particular point in time, is known as the local trajectory of the robot. As the robot makes progress, static obstacles that were not known a priori might be blocking the initial global path. Many motion controllers deal internally with this situation and generate velocities that drive the robot around obstacles, resulting in local trajectories that largely deviate from the original global path as shown at the top of Figure 2.2. An alternative is to request global path re-planning, resulting in a new collision-free path, which can be closely followed by the motion controller as shown at the bottom of Figure 2.2. The high-level component task coordination is thus in charge of harmonizing path planning and motion control to complete the navigation task.

In the remaining of this article, we will focus on motion control, and we assume a global path is available or can be requested at any time. For path planning there exist multiple algorithms, from classical A*, Rapidly-exploring random trees (RRT) [7], to Machine Learning path planning [2]. An overview of planners available in ROS2 can be found here [22].

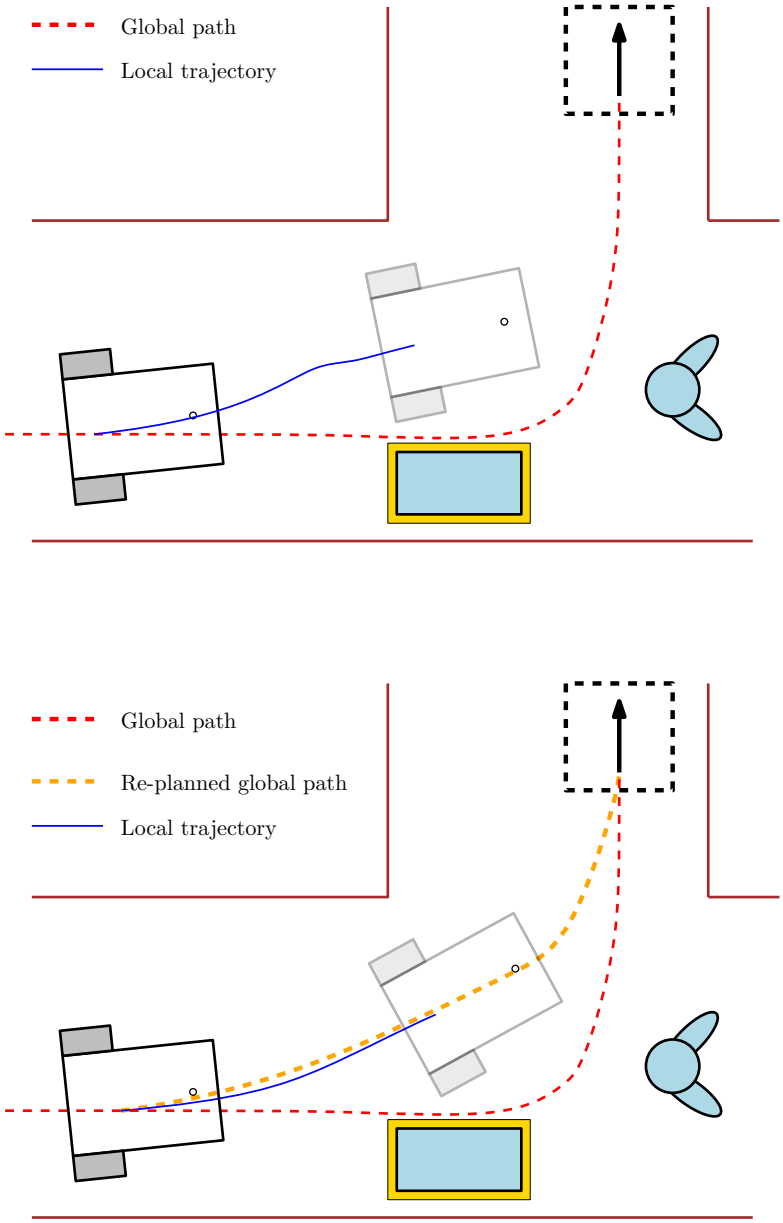


Figure 2.2: For obstacles that are not previously known, two approaches are followed: Motion controller plans a new trajectory (top), or a new global path is generated (bottom)

3 Under The Surface of ROS2 Motion Controllers

In this section we will dive into the details of the available ROS2 motion controllers [16].

3.1 DWB Motion Controller

DWB in a nutshell

The DWB motion controller, which is the successor of DWA, uses a trajectory generation and selection approach to enable a mobile robot to follow a desired global path while avoiding static and dynamic obstacles. The rate at which this generation and selection process happens is defined by the `controller_frequency` parameter. Further, the criteria for selecting a trajectory to be executed from a set of generated trajectories can be configured via user or application-specific cost functions called `critics`.

3.1.1 The Dynamic Window Approach (DWA)

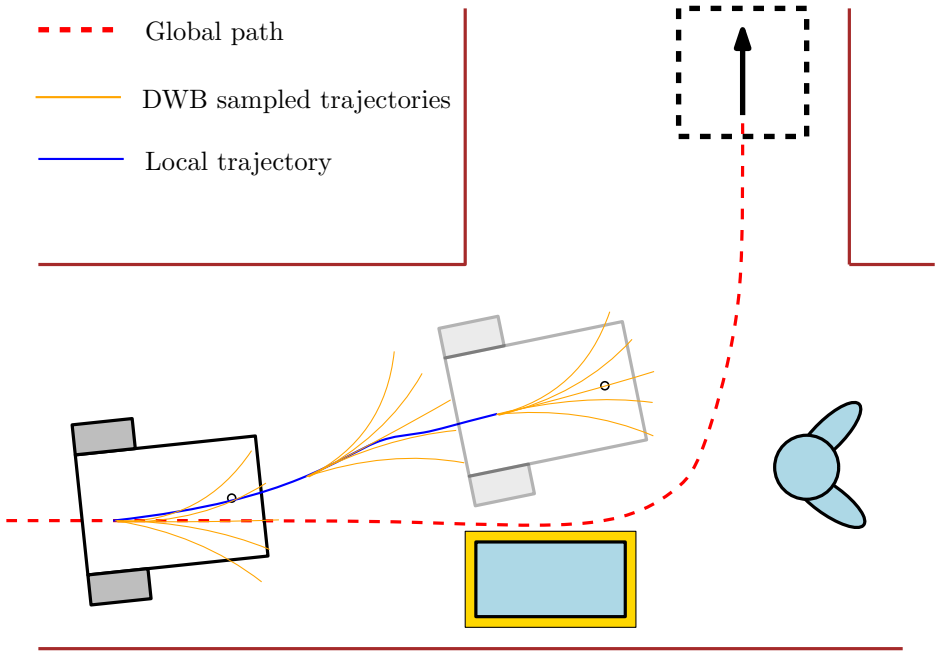


Figure 3.1: Dynamic Window Approach strategy: Discrete sampling of several velocities followed by a selection based on several criteria

The Dynamic Window Approach (DWA), proposed in [5], is a technique to generate motion commands for a robot to follow a desired global path *in the best possible manner* while avoiding dynamic obstacles. This is achieved by iterating through the following steps [1] until the robot reaches the desired goal location:

1. Discretely sample from the robot's control space
2. For each sampled control, perform a forward simulation from the robot's current state to predict what would happen if the sampled control were applied for a short period of time
3. Score each resulting trajectory from the forward simulation, using a metric that incorporates characteristics such as proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
4. Pick the highest-scoring trajectory and use the associated controls.

It is pertinent to note that, Step 2 in the above sequence is key to the performance and usability of the motion controller for a given robot and a given application. In other words, the choice of the model used to do the *forward simulation* should be as close as possible to the physical robot's movements when a given control input is applied.

In many practical uses of this motion controller in ROS, this key component is often overlooked and in turn leads to incorrect conclusions about the usability of this motion controller. To some extent, the software interface for selecting the type of the forward simulation model also contributes to the incorrect usability conclusions. This is because, the parametric configuration allows to choose between, say, a *differential drive* or an *omni drive* simulation model, but the actual models used for simulation can only be changed through code modification in the motion controller implementation. Nevertheless, this level of abstraction is sufficiently applicable to many practical prototypical applications and hence this motion controller is still used often by many users of the ROS navigation stack. However, when accurate choices between candidate trajectories are to be made, this generalization can prove to be a deterrent to some users in adopting this motion controller for their applications.

[1] Rephrased from http://wiki.ros.org/dwa_local_planner#Overview to adhere to the flow of this text.

3.1.2 Motivation for DWB

Due to its inherent ability to deal with dynamic obstacles, the DWA motion controller is typically considered for service robot applications such as home assistant robots and guidance robots at airports, museums and so forth. Further, how well the motion controller enables key task execution requirements, such as following a global path and avoiding dynamic obstacles, are decided by the scoring functions as mentioned in Step 3 in Section 3.1.1. These scoring functions are called *critics* in the software implementation in ROS. Henceforth, in this article, we will use the term critics when referring to the scoring functions. In the DWA motion controller implementation, every forward simulated trajectory gets scored by **all** the critics before a candidate trajectory is selected.

However, this is often not desirable because not all navigation applications have the same global path following requirements as imposed by the critics. In other words, using all the critics to score candidate trajectories can lead to extremely conservative candidate trajectory choices or no valid trajectory choice at all. This is a scenario that causes the robot to perform recovery behaviors in order to get a renewed estimate of its state. If this leads to a valid trajectory, the robot may continue further in following the desired global path. In the worst case, when a recovery attempt also fails due to no valid trajectory being available, a navigation failure is reported and the robot aborts its pursuit of the goal. While such a situation might be acceptable in experimental setups, it would be absolutely undesirable to have navigation and hence, overall task execution failures in real applications as mentioned earlier.

These limitations of the DWA implementation form the primary motivation for the DWB motion controller. Fundamentally, DWB is a modularized and enhanced version of the DWA motion controller incorporated via the following implementation changes:

- Configurable selection of critics
- Customizable critics without having to change the core package binaries
- Including acceleration constraints on trajectory generation

We will elaborate on each of these points in the following section. Before we proceed, it is important to reiterate that, fundamentally, the principles of the DWA motion controller are very much the same in DWB and it follows the very same sequence of steps as listed in Section 3.1.1. It is only the modularity introduced by the above-mentioned changes that distinguishes the DWA and DWB

motion controllers. Therefore, although there is no specific expansion for the letter “B” in DWB, the author of DWB has simply selected it as a natural alphabetic extension of its predecessor DWA.

3.1.3 Benefits of DWB

The most important benefit of the DWB implementation is the flexibility it brings with its configurable selection of critics. One or more critics, that score a candidate forward simulated trajectory, can be configured via the specification of the `critics` parameter in the `yaml` configuration of the navigation stack. This parameter is specified as a list of strings, where each string corresponds to a trajectory scoring function for a certain criteria. For example, a criterium to penalize trajectories that are above a certain euclidean offset distance from the global path can be enabled by adding the `PathDist` string to the `critics` parameter list. Further explanations on the different kinds of standard critics are available in the DWB GitHub documentation[2].

[2] https://github.com/locusrobotics/robot_navigation/tree/noetic/dwb_critics#dwb_critics

3.2 Regulated Pure Pursuit

Follow the Carrot!

The Regulated Pure Pursuit motion controller enables a mobile robot to track a global path by continuously moving forward a target point called *lookahead* point on incremental segments of a global path. For every lookahead point, control commands are generated so that the robot starts moving towards it and this process is repeated until the goal is reached. This controller is a variation of the classic pure pursuit approach with enhancements such as collision checking and control command regulation.

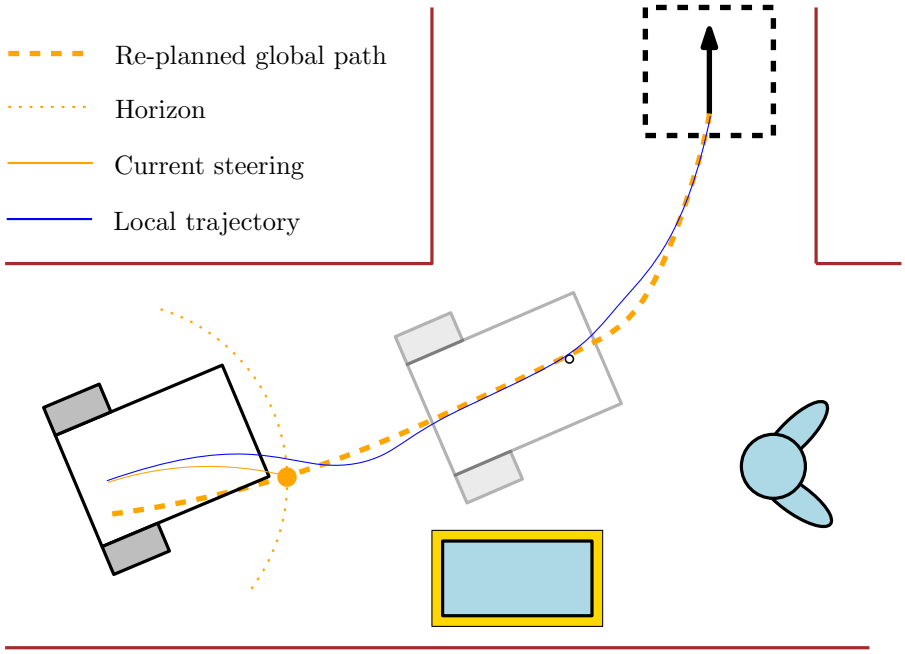


Figure 3.2: Regulated Pure Pursuit strategy: Look ahead on the path to compute carrot target and compute feedforward steering to reach it

The Regulated pure pursuit motion controller is inspired by the pure pursuit algorithm [4] and has been modified to include practical constraints such as collision with obstacles and velocity regulation during the robot's pursuit of the lookahead point. We will elaborate on the working principles of this motion controller by first starting with the basics of the Pure Pursuit algorithm.

3.2.1 Basics of Pure Pursuit

The Pure Pursuit algorithm is the core component of the motion controller. In the introductory paragraph earlier, we referred to the idea of a *lookahead* target. This is also oftentimes called a *carrot* target and typically lies on a circular horizon around the robot's control point as indicated in Figure 3.2. The radius of this circle is a parameter that can be tuned as per the application requirements. In the ROS2 implementation of this motion controller, this parameter is called `lookahead_dist`.

There are many (theoretically infinite) possibilities where this lookahead point could be around the robot. To determine the most relevant lookahead point, the global path that the robot has to follow is used as a cue. First, the robot's current pose is used to determine the nearest point on the global path based on the euclidean norm. Then, this point is used as a reference to determine the closest of the lookahead points on the circular horizon, which also lies on the global path *ahead* of the robot. This is indicated by the yellow dot in Figure 3.2. Once the appropriate lookahead point is determined, it is transformed from the global coordinate frame to the robot's local coordinate frame. Finally, this point in the local coordinate frame is used to determine the control commands to reach the lookahead point. These commands can be exactly computed based on the algebra described in Section 2.0 of [4].

3.2.2 Regulated Pure Pursuit

The Regulated Pure Pursuit (RPP) motion controller extends the basic pure pursuit algorithm to enhance its applicability to a wider scope of practical problems. As elaborated by the authors in [29], this is achieved via the introduction of the following features:

- Active collision detection
- Velocity-scaled lookahead points
- Velocity modulation when the robot approaches the desired goal

The vanilla version of the Pure Pursuit algorithm, as described in Section 3.2.1, generates control commands based on the lookahead point with the underlying assumption that there are no collisions along the way to the lookahead

point. This is generally valid as long as the lookahead horizon is not too large. Further, the availability of local costmap information in ROS opens up the possibility to quickly evaluate the path to the lookahead point for collisions. In the ROS2 implementation of this motion controller in [29], a time parameter for the maximum allowable time before a collision is introduced. In other words, a forward state projection is computed using the generated control commands from pure pursuit and the resulting robot path is evaluated for collisions. The primary motivation for this, as highlighted by the authors in [29], is to enable short local horizons within the lookahead region for evaluating the generated control commands when navigating in tightly confined spaces.

The next improvement in the RPP implementation is the possibility to include velocity-scaled lookahead points. The fundamental idea here is that when a robot is navigating at higher speeds, it is logical to have a larger lookahead horizon and adapt the motion commands based on the larger range. In essence, this feature enables a user to have variable lookahead distances and in turn influence the control command outputs of the underlying pure pursuit algorithm. This is because the control commands are a function of the lookahead distance. For example, this feature can enable a robot to speed up towards its goal when the coast is clear in the visible horizon of the onboard sensors, which are usually larger than a fixed lookahead distance.

Finally, the last improvement in RPP is the feature of slowing down on approach to the goal. This is achieved by simply checking for the existence of lookahead points on the global path. This relies on the basic idea that a lookahead point beyond the goal point cannot exist. Hence, by monitoring an offset error to the goal point from the robot's current position, the control commands to the robot can be proportionally reduced to ensure a slow approach to the goal position on the global path before the robot comes to a halt.

It is important to note that the RPP motion controller only reports a collision if it is detected during the collision-checking process. However, it does not generate an alternative path around the obstacle and depends on the global planner to regenerate a path around the obstacles. Further, if the lookahead horizon is small, when operating in tightly confined spaces, the generated control commands can be highly sensitive to noise because the generated control commands always intend to reach the lookahead point. In other words, for short lookahead horizons, the control commands tend to change rapidly and need additional processing before practical use.

3.3 Path Tracking PID

High tracking performance controller

This motion controller addresses the challenge of accurately following a predetermined path, which is also known as tracking control. Nobleo developed a high-performance path-tracking algorithm for mobile robots, which uses the feedback and feedforward principles of control theory.

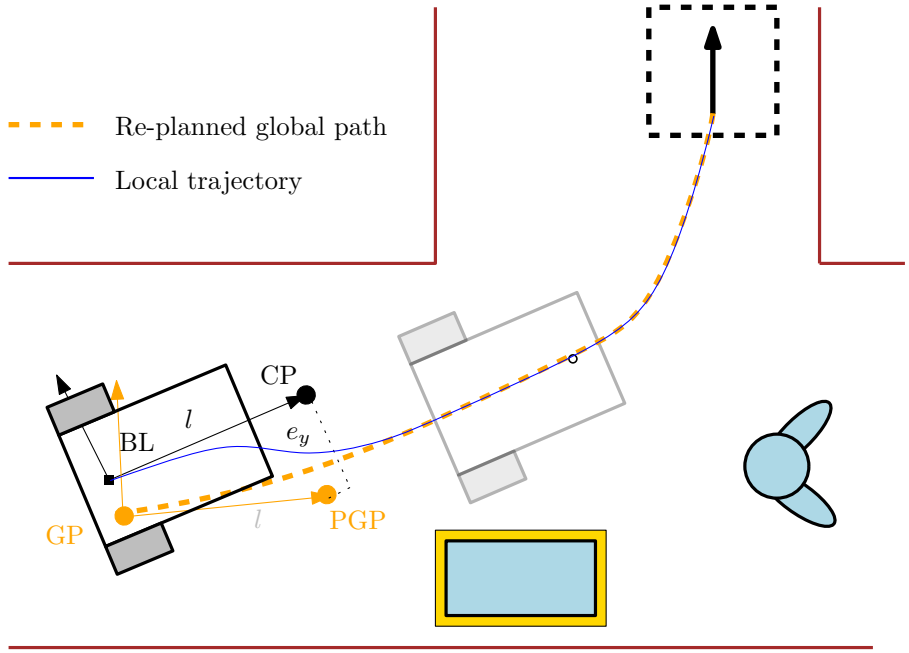


Figure 3.3: Path Tracking PID strategy: Compute carrot target by projecting the closest pose on the path (i.e. the Projected Global Path (PGP)). Velocities are computed using a PID controller with the error between the Control Point (CP) and the PGP

In some applications, like precision agriculture and pallet picking, it is desirable to accurately follow a predetermined path. Prior to the release of the Path Tracking PID by Nobleo [12], there was no open-source motion controller with the required level of performance. The essence of the Path Tracking PID is depicted in Figure 3.3. The concept relies on two aspects: projection of the global path, and accurate control of a point ahead of the robot, known as a carrot point.

The idea behind the global path projection is to compute a path that if followed accurately with a carrot point or control point (CP), will result in the base link (BL) of the robot, defined at the rotation point of the robot, following the original global path. At each update step, instead of computing the projected path, the closest point from the base link to the path is found (i.e. the Global Pose (GP)) and a Projected Global Pose is computed (PGP). The carrot point CP is simply calculated by projecting the base link in the direction of the orientation of the robot.

The next step is to generate velocity and steering commands such that as time progresses the CP gets close to the PGP, and ideally their location stays together, i.e. the control error e_y remains small as the robot progresses along the path. To be able to achieve this, the Path Tracking PID uses concepts applied in the control of high-precision equipment, mainly the combination of feedback and feedforward control. The corresponding architecture is shown in Figure 3.4.

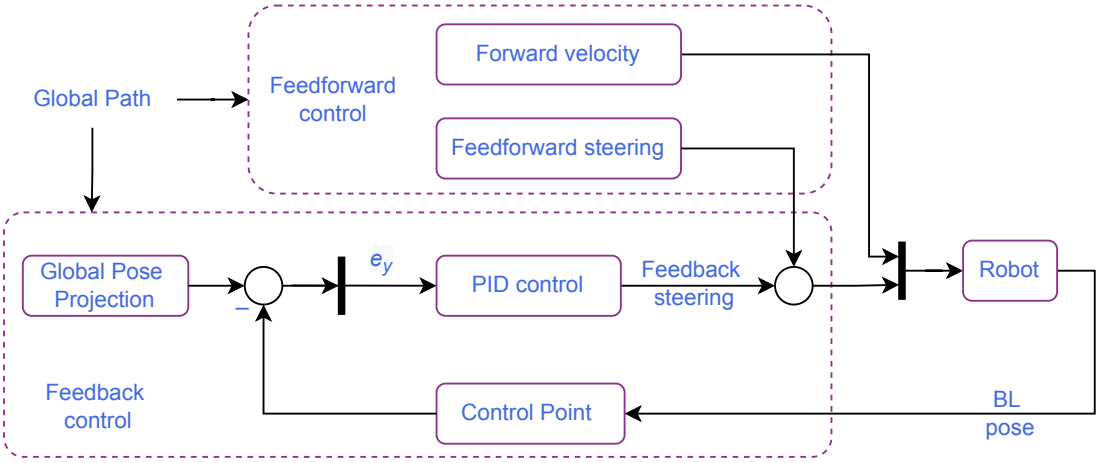


Figure 3.4: Path Tracking PID's architecture uses a combination of feedforward and feedback control. Forward velocities are purely computed as feedforward and steering commands, a combination of feedback and feedforward.

In feedback control, the measurement of the robot's base link with respect to the path is used to compute a steering command that makes the robot get closer to it. Specifically, PID control is used which stands for Proportional, Integral, Derivative control. The Proportional part uses the current position error e_y, the derivative part uses the changes in time of e_y and the integral error uses the summations of all values of e_y in the past. Without going into too much depth in control theory, assuming the robot moves at a constant velocity, when the

path is a straight line and with proper tuning of the PID controller parameters, it is guaranteed that a PID controller can drive the control error e_y towards zero. In practice, however, the path contains curves, which will make the PGP move sideways as seen from the robot's perspective. This will act as a disturbance to the PID controller, which makes that the control error does not tend to zero, and therefore, the robot will not accurately follow the desired path. To solve this issue, the feedforward control concept is used.

In feedforward control, preliminary knowledge of the path and the desired trajectory is used to generate velocity and steering commands for the vehicle. As illustrated in Figure 3.4, two components of feedforward are used, the forward velocity and the feedforward steering.

The forward velocity of the robot is generated as follows. During the first phase, the velocity is increased at a constant acceleration towards the desired target velocity. During the second phase, the target velocity is kept constant until the third phase is reached, where the velocity is increased or decreased at a constant rate towards the desired velocity at the end of the path (i.e. the end velocity) which can be non-zero. Note that the second phase might be skipped when the path is too short and thus only the (de)acceleration phases are performed. In this procedure, the position of the robot is monitored to know when to trigger the last phase.

The feedforward steering uses information of the path's curvature to generate a steering command. It basically uses the same principle as the pure pursuit, but instead of using the current robot's pose towards a target ahead in the path, the Global Pose (i.e. closest pose on the path) is used to calculate the steering that would drive the robot towards the next pose in the path.

Because measurements are used indirectly in the feedforward calculation, the resulting feedforward signals are commonly noise-free and smooth. It is important to mention that the generated path needs to be sufficiently smooth for the Path Tracking PID to achieve high accuracy. Discontinuities and high values in the curvature will be reflected in the generated velocity and steering commands, which in practice cannot be executed by the robot.

To conclude, if the path is properly designed, one can expect that the path tracking error will be small. This controller has been applied in high-precision agriculture applications, where a tractor performs work on the land [1] and tracking errors of less than five centimeters have been achieved.

3.4 Timed Elastic Bands (TEB)

A non-linear optimization based controller

The Timed Elastic Band (TEB) planner formulates motion control as a non-linear optimization problem. Such an approach has been already explored in the scientific community. However, the TEB is the only planner that is publicly available in ROS/ROS2. A key difference of TEB with respect to other methods is that it can efficiently solve the needed equations, making it usable for mobile robots without the need for powerful computers.

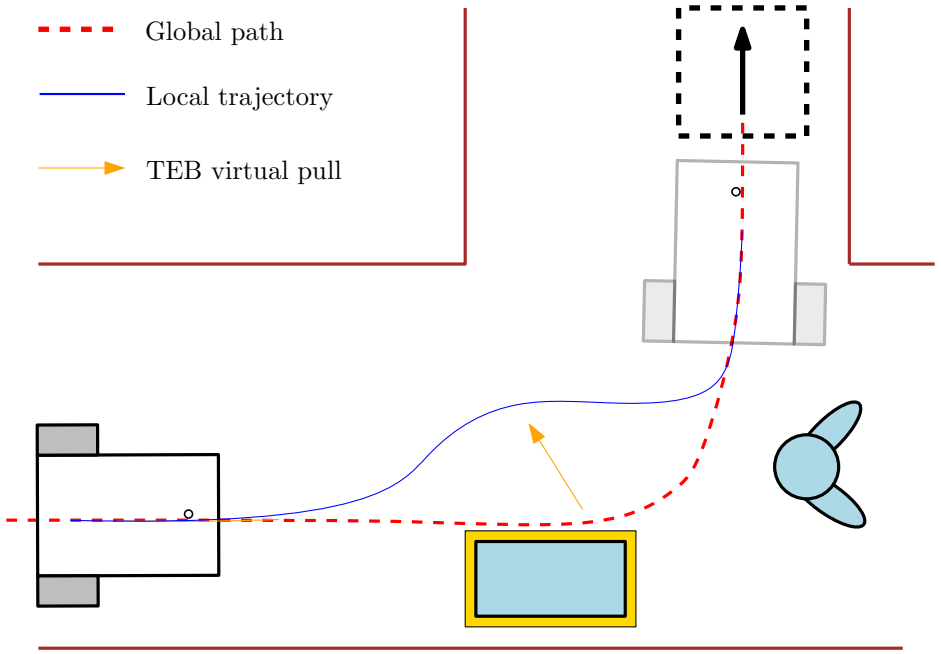


Figure 3.5: Timed Elastic Bands strategy: An optimization problem is formulated such that virtual springs with time stamps are added along the global path. The virtual spring allows the path to be elastic and malleable to avoid obstacles.

The TEB local planner [15] is the only motion controller available in ROS/ROS2 that is based on solving a non-linear optimization problem. Thus, instead of relying on a specific strategy or function to generate velocity commands, TEB relies on solving an optimization problem with constraints. A simplified representation of the formulated optimization problem is illustrated in Figure 3.5. Virtual

springs are added at certain locations along the path and time stamps are added to them. Far from obstacles, the virtual springs are not active, and the resolved trajectory would be very close to the original path. When the robot approaches an obstacle, the virtual springs are activated, and the trajectory is “pulled” away from the original path.

What makes TEB successful, is the way the non-linear optimization problem is formulated to be solved with relatively few resources, and therefore accessible to be used in mobile robots with low to medium onboard compute. The key aspects of TEB problem formulation are the following. The internal control variable is formulated as a sequence of arcs, for which the radius can be limited to account for kinematic constraints. As shown in Figure 3.6, the robot footprint can be described in different ways: as a distance to a line segment, two circles, or a polygon. Especially the first two options can reduce the number of calculations drastically with respect to a full polygon. In addition, not all poses on the trajectory are affected when there are obstacles, but only those closest to them, which can also be tuned by the user. In this way, the user can trade off between accuracy, robustness and computation complexity.

The other important aspect of TEB is the way the optimization problem is solved. Across the different versions of the TEB core developed in the last years [27], [26], [25], the commonality is that the original non-linear optimization problem with constraints is approximated as a non-linear Least-Squares optimization problem without constraints. In particular, they use the graph optimization framework g2o [8] which implements a highly efficient solver. This however comes with a price as the approximation might not describe the original problem correctly, leading to incorrect results.

Finally, it is important to mention that as with any non-linear optimization problem, there are known issues while solving such types of problems. Firstly, a global minimum is not guaranteed, or even to find a feasible solution. Tuning properly can help to obtain good results and can prevent getting trapped in local minima. However, the problem solved by TEB is still an approximation of the original one, so correctness is not guaranteed. Moreover, the amount of parameters to be tuned in TEB is quite long [11], and tuning can be quite non-intuitive due to the influence the parameters have on each other.

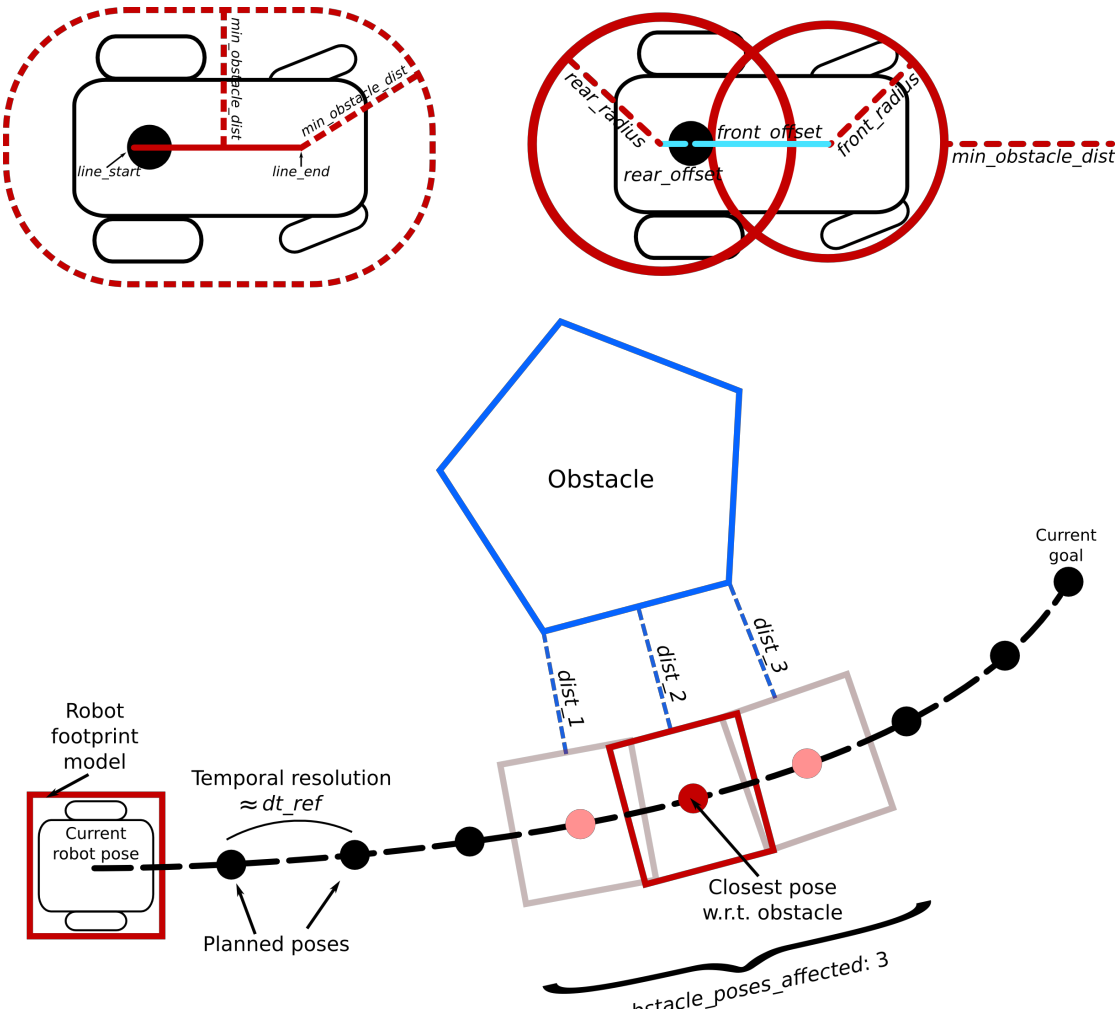


Figure 3.6: TEB formulation key aspects: Footprint can be expressed as a distance to a line segment, two circles, or a polygon. Also, only a subset of the trajectory is considered in the optimization. Images taken from [6].

4 Motion Controllers Benchmark

Benchmarking

In this section, we will compare the different ROS2 motion controllers. To make a valid and realistic comparison, we selected two robots used in real-life applications: a care robot and a logistics robot. We also used different scenarios to compare the performance of the different controllers.

4.1 Benchmark description

4.1.1 Robot examples

For our comparison, we have selected two robots that are used in real-life scenarios: a care robot and a logistics robot. They have different sizes and placements of the rotation point, which highlights the challenges in motion control.

SARA

SARA is a robot brought to the market by SARA robotics[13], and it is used to assist tasks in the healthcare sector. In the past, Nobleo has collaborated with SARA robotics to bring new functionalities to their product [28]. In the context of this comparison, SARA represents a robot whose footprint is relatively small with respect to its environment. Though SARA is holonomic (i.e. it can also move sideways) to generalize it to an industrial robot, we will treat it as a differential drive robot. This is because, differential drive design is commonplace in industrial AMRs for applications like logistics, maintenance and inspection.

Figure 4.1: SARA robot [13] is used to assist in healthcare tasks.

IDA

IDA was initially a manually driven pallet truck that was made fully autonomous by Nobleo [10]. IDA is capable of autonomously navigating in a warehouse environment and loading and unloading pallets. The rotation point of IDA is located close to the back of the robot, and in the front it has a steering wheel. In this comparison, we will assume the steering wheel can rotate very fast to make a fair comparison with a differential drive.

Figure 4.2: IDA is an autonomous pallet truck, whose autonomy feature as develop by Nobleo [10].

4.1.2 Test description

We designed three test scenarios shown in Figure 4.3 to benchmark the different controllers.

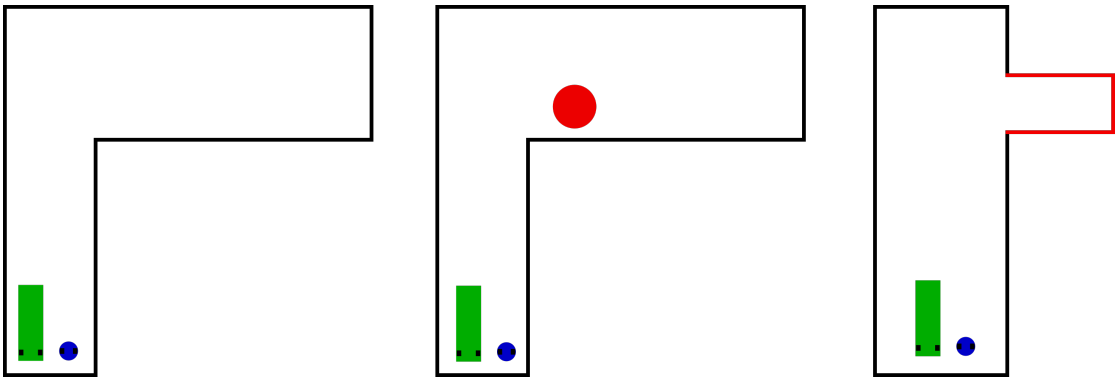


Figure 4.3: Test scenarios: left, turn from hallway to hallway; center, turn from hallway to hallway with an unknown obstacle; right, parking in a tight space. For size comparison, the footprints of SARA and IDA are shown as a blue circle and a green polygon respectively.

In the first test on the left, the robot needs to move along a hall and then turn right to continue heading towards the target pose. With this test, we want to check the basic ability of different controllers to drive safely. In the second test in the middle, a previously unseen object is placed around the corner. Here, we will test the ability of the motion controller to deal with previously unknown obstacles that are placed along the original path. Finally, in the third test on the right, the robot needs to park in a tight space, depicted in red. Here we are testing the ability of the motion controller to accurately follow the path and produce accurate movements in a tight environment. Footprints of both robots are also shown in relation to the environment. It is expected that navigation with an elongated robot such as IDA is more challenging compared to navigation with a circular robot.

A single test is composed of a combination of a test scenario, a robot, and a motion controller. In total, three test scenarios, two robots, and four controllers, adding up to twenty-four tests are performed. For the tests we make use of the Webots simulator [32] since it allows for easy (re)spawning of maps, robots and obstacles. The test procedure goes as follows: A map, a robot and an obstacle (when needed) are loaded into Webots. After launching the navigation stack in

ROS2, a (global) plan is requested from the SmacPlanner [24] with the desired target position. The SmacPlanner was selected due to its ability to handle a wide range of robots and generate smooth paths. Subsequently, the corresponding motion controller of the current test is invoked and navigation starts. The tests have a maximum execution time, and when it is exceeded, the test is deemed unsuccessful.

4.1.3 Test results

The test results for the SARA robot are shown in Figure 4.4. As expected all tests for turning succeeded and there are no major differences in execution time. Regarding the test with the obstacle, it was already expected that the path tracking PID and regulated pure pursuit would fail, since both do not circumvent unforeseen obstacles. The TEB local planner succeeds since it is quite flexible and a circular footprint is easily handled by its optimization approach. DWB is also able to drive the robot safely around the obstacle, although it requires more execution time because DWB does not perform exhaustive optimization on candidate trajectories. Finally, all controllers succeed in the parking situation. This is not surprising since the footprint of SARA is small relative to the parking space.

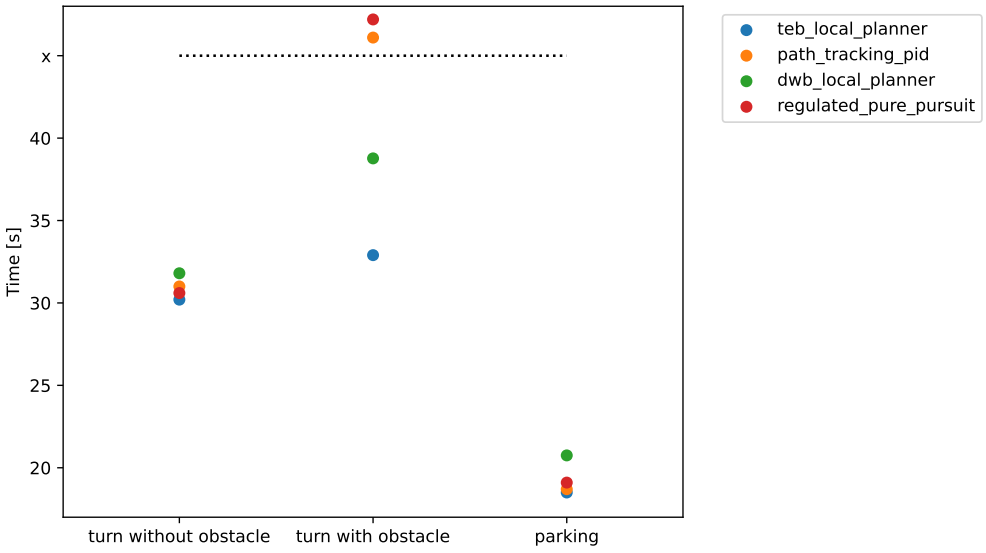


Figure 4.4: SARA test results. Results above the horizontal dashed line mean that the particular test did not succeed.

The test results for the IDA robot are shown in Figure 4.5. Similar to SARA, all tests for turning succeeded without major differences in execution time. Regarding the test with the unforeseen obstacle, TEB and DWB succeeded, showing that DWB can also be used with large footprints given there is enough space for maneuvering, especially DWB required additional tuning effort to make it work in this scenario. Finally, for the parking maneuver with IDA, only TEB and path tracking PID succeed, the former due to its flexibility and the latter due to its accuracy. We observed DWB would drive the robot close to the path but not very accurately, and IDA would hit a corner right at the entrance of the parking spot. The regulated pure pursuit failed because by design it follows a point ahead in the path, which makes the robot "cut corners", and therefore IDA also got stuck entering the parking spot.

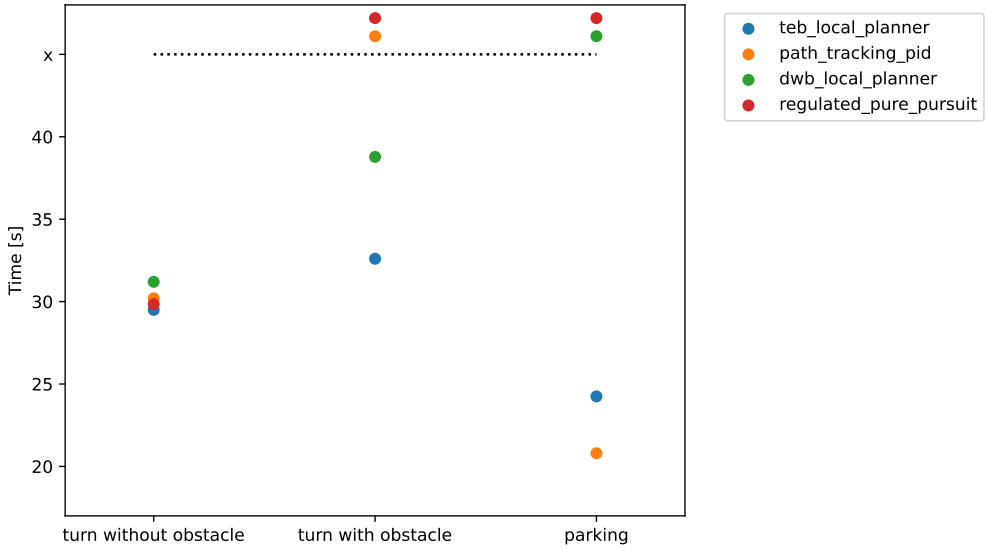


Figure 4.5: IDA test results. Results above the horizontal dashed line mean that particular test did not succeed.

The results show that TEB, unlike the other planners, has great flexibility to operate under different scenarios, and different footprints. However, we noticed that during the test executions TEB at times was facing numerical issues, and the solution trajectory was changing drastically from one sample time to the other. This behavior was mainly observed during the tests with IDA, which can be caused by the high non-linearities that a non-symmetric footprint brings into the optimization problem. Path tracking PID cannot cope with obstacles but is able to accurately follow the original global plan. Also, its algorithm is deterministic and it produces reliable results.

4.2 Selection Guidelines

From the different controller concepts and observations made during testing, we got some insights to score each controller based on certain criteria. Therefore, we present a concise selection guideline in Table 4.1.

Criteria	**DWB**	**RPP**	**PTPID**	**TEB**
Static known environments	+	+	+	+
Unknown obstacles	++	-	-	++
Small/symmetric footprint	+	+	+	+
Large/asymmetric footprint	+	-	++	+
Accuracy	+	-	++	+
Robustness/Reproducibility	+	+	++	-
Computational simplicity	+	++	++	- -

Table 4.1: ROS2 motion controller selection guideline for Dynamic Window Approach B (**DWB**), Regulated Pure Pursuit (**RPP**), Path Tracking PID (**PTPID**) and Time Elactic Bands (**TEB**).

In the criteria static known obstacles, all controllers have a similar score since they fulfill the basic A to B navigation requirement. For unknown obstacles, TEB scores the best, followed by DWB which performs well if some tuning effort is put into it. Regarding footprint size, all controllers scored well for small footprints. For large footprints, meaning maneuvering in tight spaces, PTPID performs well given the global path is correct and smooth, and TEB offers the greatest flexibility in this respect. For accuracy, PTPID is the winner, and only RPP scores low because of its tendency to “cut corners”. PTPID also scores the best in robustness because of its reproducible results, and TEB the lowest because it can face numerical issues while solving the non-linear optimization problem. Finally, regarding computational simplicity, PTPID and RPP score the best because their control law is simple, straightforward, and does not require extensive calculations. On the other hand, DWB already requires more calculations to sample multiple velocity pair candidates, and TEB scores the lowest due to the high demand for processing power.

Looking at each controller column, we can see DWB has a good balance among the selected criteria, and it was consistently providing obstacle avoidance behavior. RPP can perform well for robots with small footprints in a known

environment and is computationally simpler than DWB. PTPID specializes in accuracy and reproducibility, and finally, TEB offers great flexibility but it requires high computation power and it can lack robustness in specific situations.

Before using this table, one should score the relevant criteria of the application that the robot is intended for. As an example, for an autonomous tractor, accuracy and robustness are very important. Circumventing obstacles on the other hand is less relevant. Of course, avoiding collision along the global path is a must that is a basic feature of any controller. Computational simplicity is also important because many hardware elements need to be handled simultaneously on top of robot navigation. Taking these elements into account, one can conclude that Path Tracking PID is a good choice for such an application. Another example is a surveillance robot in a shopping mall. The robot has a relatively small footprint with respect to the environment, accuracy is not relevant but dealing with unknown obstacles is relevant. These aspects then make DWB a good candidate for this application.

Conclusion

In this paper, we have taken a close look at the navigation software for mobile robots provided by ROS2. We focused on the motion controller, which is the module that is ultimately responsible for generating the velocity commands that drive the robot through the environment. We first explained the underlying mechanisms of four motion controllers available in ROS2: DWB, Regulated Pure Pursuit (RPP), Path Tracking PID (PTPID) and Timed Elastic Bands (TEB). These controllers were tested using two robots with commercial applications in three different scenarios. The first robot, SARA, used in healthcare tasks, has a small footprint with respect to the environment. The second robot, IDA, an autonomous pallet truck, has a large and asymmetric footprint. The test scenarios were focused on basic navigation, dealing with previously unknown obstacles, and navigating in a tight environment.

In general, the different controllers have different strong suits with respect to the evaluation criteria, making them complementary. DWB has a balanced score which makes it suitable for a broad range of applications. RPP is computationally efficient but not very accurate. PTPID was designed for accuracy and consistency and performs the best at it. TEB is very flexible but its lack of robustness makes it less suitable for some industrial applications. Finally, a proper selection of motion controller greatly depends on the application requirements.

Bibliography

[1] AgXeed.
AgXeed.
2022.
URL: https://www.agxeed.com/ (visited on 11/25/2022).

[2] Szilárd Aradi.
"Survey of deep reinforcement learning for motion planning of autonomous vehicles."
In: *IEEE Transactions on Intelligent Transportation Systems* (2020).

[3] Control Automation Community.
The Evolution of Autonomous Mobile Robots.
2020.
URL: https://control.com/technical-articles/the-evolution-of-autonomous-mobile-robots/ (visited on 11/25/2022).

[4] R. Craig-Coulter.
Implementation of the Pure Pursuit Path tracking Algorithm.
1992.
URL: https://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/MRIS/coulter%5C_r%5C_craig%5C_1992%5C_1.pdf (visited on 12/11/2022).

[5] S. Thrun D. Fox W. Burgard.
"The Dynamic Window Approach to Collision Avoidance."
In: *IEEE Robotics and Automation Magazine* 4 (1997), pp. 23–33.

[6] TEB footprint.
TEB footprint.
2016.
URL: http://wiki.ros.org/teb_local_planner/Tutorials/Obstacle%20Avoidance%20and%20Robot%20Footprint%20Model (visited on 11/25/2022).

[7] Lydia E. Kavraki and Steven M. LaValle.
"Motion Planning."

In:
Springer Handbook of Robotics.
Ed. by Bruno Siciliano and Oussama Khatib.
Cham: Springer International Publishing, 2016,
Pp. 139–162.
ISBN: 978-3-319-32552-1.
DOI: 10.1007/978-3-319-32552-1_7.
URL: https://doi.org/10.1007/978-3-319-32552-1_7.

[8] Rainer Kümmerle et al.
"g 2 o: A general framework for graph optimization."
In: *2011 IEEE International Conference on Robotics and Automation*.
IEEE. 2011,
Pp. 3607–3613.

[9] Pascal Morin and Claude Samson.
"Motion Control of Wheeled Mobile Robots."
In:
Springer Handbook of Robotics.
Ed. by Bruno Siciliano and Oussama Khatib.
Berlin, Heidelberg: Springer Berlin Heidelberg, 2008,
Pp. 799–826.
ISBN: 978-3-540-30301-5.
DOI: 10.1007/978-3-540-30301-5_35.
URL: https://doi.org/10.1007/978-3-540-30301-5_35.

[10] Nobleo.
IDA Autonomous pallet truck.
2022.
URL: https://nobleo-technology.nl/project/autonomous-pallet-truck/ (visited on 11/25/2022).

[11] TEB parameters.
TEB parameters.
2021.
URL: https://github.com/rst-tu-dortmund/teb_local_planner/blob/galactic/teb_local_planner/cfg/TebLocalPlannerReconfigure.cfg (visited on 11/25/2022).

[12] Nobleo Path Tracking PID.
ROS Conference 2011: Path Tracking PID.
2021.
URL: https://vimeo.com/635607300#t=1913s (visited on 11/25/2022).

[13] SARA robotics.

SARA robotics.
2022.
URL: https://sara-robotics.com/ (visited on 11/25/2022).

[14] ROS.
Robot Operating System.
2022.
URL: https://www.ros.org/ (visited on 11/25/2022).

[15] TEB ROS.
TEB ROS.
2020.
URL: http://wiki.ros.org/teb_local_planner (visited on 11/25/2022).

[16] ROS2.
Nav2.
2020.
URL: https://navigation.ros.org/ (visited on 11/25/2022).

[17] ROS2.
Nav2 Architecture.
2020.
URL: https://navigation.ros.org/_images/nav2_architecture.png (visited on 11/25/2022).

[18] ROS2.
Nav2 Behavioral Trees.
2020.
URL: https://navigation.ros.org/concepts/index.html#behavior-trees (visited on 11/25/2022).

[19] ROS2.
Nav2 environment representation.
2020.
URL: https://navigation.ros.org/concepts/index.html#environmental-representation (visited on 11/25/2022).

[20] ROS2.
Nav2 motion control.
2020.
URL: https://navigation.ros.org/concepts/index.html#controllers (visited on 11/25/2022).

[21] ROS2.
Nav2 motion plan.
2020.

URL: https://navigation.ros.org/concepts/index.html#planners (visited on 11/25/2022).

[22] ROS2.
Nav2 planners.
2020.
URL: https://navigation.ros.org/plugins/index.html#planners (visited on 11/25/2022).

[23] ROS2.
ROS2 documentation.
2022.
URL: https://docs.ros.org/en/humble/index.html (visited on 11/25/2022).

[24] ROS2.
Smac Planner.
2022.
URL: https://navigation.ros.org/configuration/packages/configuring-smac-planner.html (visited on 11/25/2022).

[25] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram.
"Kinodynamic trajectory optimization and control for car-like robots."
In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
IEEE. 2017,
Pp. 5681–5686.

[26] Christoph Rösmann et al.
"Efficient trajectory optimization using a sparse model."
In: *2013 European Conference on Mobile Robots*.
IEEE. 2013,
Pp. 138–143.

[27] Christoph Rösmann et al.
"Trajectory modification considering dynamic constraints of autonomous robots."
In: *ROBOTIK 2012; 7th German Conference on Robotics*.
VDE. 2012,
Pp. 1–6.

[28] Nobleo SARA.
SARA Nobleo collaboration.
2022.
URL: https://nobleo-technology.nl/project/care-robot-sara/ (visited on 11/25/2022).

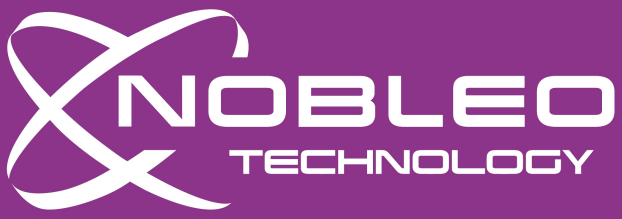
[29] S. Singh and S. Macenski.
Nav2 Regulated Pure Pursuit Controller.
2021.
URL: `https://github.com/ros-planning/navigation2/tree/main/nav2%5C_regulated%5C_pure%5C_pursuit%5C_controller` (visited on 12/11/2022).

[30] University of Stuttgart.
Robotics: Path Planning.
2014.
URL: `https://therisingsea.org/notes/FoundationsForCategoryTheory.pdf` (visited on 11/25/2022).

[31] Günter Ullrich and Thomas Albrecht.
"History of Automated Guided Vehicle Systems."
In: *Automated guided vehicle systems*.
Springer, 2023,
Pp. 1–26.

[32] Webots.
http://www.cyberbotics.com.
Ed. by Cyberbotics Ltd.
Open-source Mobile Robot Simulation Software.
URL: `http://www.cyberbotics.com`.

[33] Wikipedia.
Ackermann Steering Geometry.
2022.
URL: `https://en.wikipedia.org/wiki/Ackermann_steering_geometry` (visited on 11/25/2022).


NOBLEO
TECHNOLOGY