ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)

https://doi.org/10.17559/TV-20220823131848

Original scientific paper

Implementation of Trajectory Planning Algorithms for Track Serving Mobile Robot in ROS 2 Ecosystem

Norbert BOROS*, Gábor KALLÓS, Áron BALLAGI

Abstract: In this paper, our goal is to present the autonomous cone placing robot developed at SzéchenyiIstván University (Győr, Hungary) and the main steps and parts of its design and preparation. Within this, rather complex task-sequence, we discuss the logic of software operation (embedded in the ROS 2 ecosystem), the main issues of environmental representation and we focus especially on the trajectory planner part of the entire system. The implemented algorithms-including our own innovative ideas are Dijkstra, *A**, Hybrid *A**, DWA and Elastic band.

Keywords: Cone placing robot; digital twin; elastic band algorithm; hybrid *A** algorithm; ROS 2 system; trajectory

1 INTRODUCTION, RELATED WORK

In this paper, we present a task that is part of a research related to the application of mobile robots, which is a very popular research field today.

By mobile robot, we mean an automated vehicle moving on the ground that can move freely from its current position to a specified target position or follow a predefined trajectory. It does this quasi-independently. These devices have modern computing and management systems, however, the concept is much older. As early as the second half of the 1960s, mobile robots were developed, but in a very embryotic manner for today's eyes ("Beast," Johns Hopkins University). A similar development was made a little later at Stanford University, and the "Stanford Cart" device equipped with a camera was already capable of tracking a white line. In the 1970s and 1980s, a number of external conditions gave a major impetus to the development of (partially) autonomously displaceable devices (industry, space, health).

The scope of the tools has become wide, including but not limited to: exploration of hard-to-reach or dangerous places (other celestial bodies, volcanoes), execution of processes requiring extreme precision, working in extreme environments or with hazardous substances, destruction of enemy objects (intelligent torpedoes). In the case of a Martian probe, for example, self-determination is obviously required, as it takes minutes for the images transmitted by the probe to be taken by ground controllers and sent any response to them [1].

Largely due to automotive applications, the concept of the autonomous vehicle ("self-driving car") was precisely formulated as early as the 1980s [2]. However, the limitations of technological development at the time did not yet allow for a truly realistic implementation. At that time, the acceptable goal could only be some modest support provided to the driver, mainly by road detection and tracking, map analysis relatively rudimentary in our current perspective.

Moreover, such research has long been relatively limited due to the difficult availability and cost of the necessary devices.

The first truly successful application was the warning system developed for unintentional lane departure in the mid-to-late 1990s. Assistance in automatic parking was also raised at an early stage, and automatic driving support to save fuel. In addition to the self-intelligent behaviour of vehicles, the study of collaborative intelligent behaviour has also been an important part of research (cooperative management, better throughput of vehicle fleets, traffic jam prevention, etc.) [3].

The range of investigations has been steadily expanding in recent decades, and the areas have become intertwined. Both in the field of mobile robots and in self-driving vehicles, the recognition of the environment, the detection of the movement of the device and the design of the trajectory have become very important research topics.

At our university, such research began under the guidance of Budapest and local researchers from the beginning of the 2000 (J. Bokor, L. Palkovics, Á. Ballagi). Several important directions (profiles) have emerged, among them the cooperative cooperation (fleet), but intensive research has been and is being carried out on the control of devices (motion planning and detection, etc.).

It should be emphasized that these developments are also important and promising from the point of view of the automotive industry (basic research, even in our wider region Northern Transdanubia, Hungary).

In this article, our specific problem is related to coning the road surface (cone packing). This is a monotonous and time-consuming work for humans, requires a high degree of precision, an intensive attention and can also be dangerous in favour of replacing the traditional and still almost exclusively used manual implementation with a mechanical force. So automated task solving is obviously important and current.

With the infrastructure available to us (exact description of the robot: see later), we want to provide a mechanical solution to this problem. The specific task, then, is to navigate a fully autonomous track robot to cone on test tracks in the automotive industry and later to collect the deposited cones. (The task of the operator is to plan the course and prepare it at a certain level, and also certain direct interventions in the robot's activities are possible.)

We note that roughly 10 people worked on the entire project for months, and work is still underway (hardware and also software in three teams separate machine vision, robotic arm, and platform control).

Our small team worked in the field of navigation, environmental representation, and route planning, with the algorithms presented in detail later. This field of research regarding mobile autonomous robots was already very popular from the end of the 1990 [4-7], but naturally, it is still so today. A nicely structured route planning summary can be read in [8].

The current stage of mobile robotics development is characterized by the satisfaction of the demand for more and more complete autonomy. We can consider the operation fully autonomous if the robot is able to perform working tasks without any external interventions. This is an exciting research field with many new results, primarily in control engineering and sensor technics. [9] Autonomous robot control is significantly based on artificial intelligence, mainly in handling uncertainty and processing environmental information. [10-12]

However, in practice, robots operate without complete autonomy, communicate to others and use external control. This is primarily necessary for safety reasons and due to legal directives.

The workability of a mobile robot is significantly influenced by its maneuverability, and very special omnidirectional solutions have been developed for this aim. [13] In the case of our mobile platform, agility is not critical, there is time to perform complicated maneuvers.

Among the latest interesting results more closely related to us we mention the article [14], in which the authors present a cone application. Unlike ours, however, this is not a completely autonomous implementation, because a car delivers the cones, and the cone-placing robot is built into it. The positioning error is max. 2 cm.

Naturally, our work also uses the trends presented above.

The main novelty of our work is that, according to our knowledge, no-one has carried out such a test with an autonomous cone-placing mobile robot (in fact: no one has even made such a robot before!), and additionally, our objectives include several unique, novel ideas that we were able to implement fully or largely (ecosystem, platform-independent solution, modified-improved algorithms, etc.) these will be presented one by one later.

2 DETAILED TASK SPECIFICATION

In this chapter, we will introduce the operation of the robot (and other important subtasks to be performed) during the execution of the coning task. We note that the more important details of the operation (physical, logical and software solutions) will be discussed later to the extent necessary for us.

The sequence of activities is as follows:

- The operator determines the nature of the task (cone on or off) and then marks the specified area on the track map (cone points can also be selected from a database).
- The map management system transmits the coordinates of the cone locations to the route planner.
- The trajectory planning system asks the current position of the robot.
- The route planning system sorts the cones positions to be "visited" according to the position of the robot, creates the trajectory and sends it to the robot.
- The robot starts to perform the task;
- If the robot encounters an (unexpected) obstacle, it bypasses it and signals it to the trajectory planning system (possible redesign).
- The operator monitors the operation of the robot (visual connection), tracks the position of the robot on the map and intervenes if necessary (jamming, possible dangerous situation).
- The robot returns to the specified starting position after completing the task.

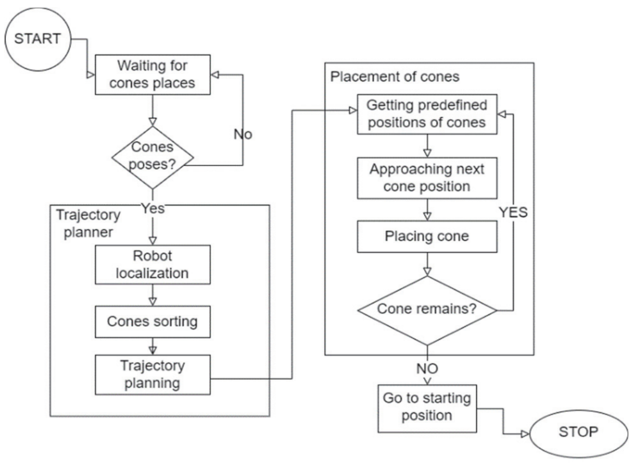


Figure 1 The logical operation of the cone-placing robot: task execution sketch, interactions

Successful implementation of the above sequence of activities requires reliable interoperability of many subsystems. Of these, the focus of the implementation is on the following tasks:

- Localization: determining the position of the robot, placed on an available map of the work area.
- Motion and trajectory planning: further motion (traversal) must be planned from the current position of the robot.
- Object and obstacle detection: the robot must detect cones (objects to be treated) and other obstacles based on the camera image (these must also be placed in the local framework for later activity).
- Manipulator handling: motion manipulation is also required for the manipulator to intervene successfully.

In the present study we mainly deal with the software control of the platform, and within this we focus primarily on the development of trajectory design algorithms, and the implementation of the route planning system, respectively.

3 CONING ROBOT: ARCHITECTURE, SOFTWARE STRUCTURE AND OPERATION

A prototype of the cone-placing/picking robot has previously been completed and the research team has been working on life-like testing on the ZalaZone test track. Details of this can be found in the article [15].

3.1 Physical Architecture

The coning robot is physically an assembly of a mobile robot platform and a robot manipulator (arm) complete with sensors. The mobile robot platform is driven by two driven wheel differential drives. An anthropomorphic

(6 degree of turn) robotic arm is attached to the platform. The latter is the primary intervener.

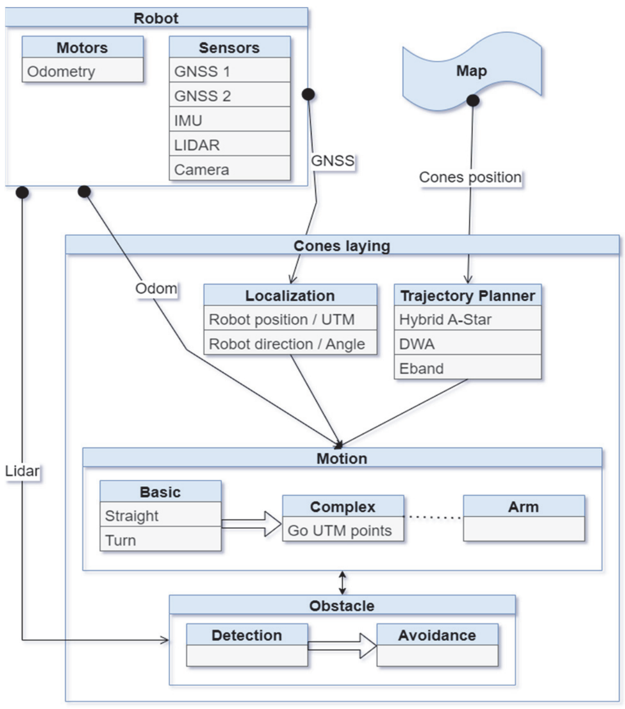


Figure 2 Software operation of the cone-placing robot: system elements, modules, tasks

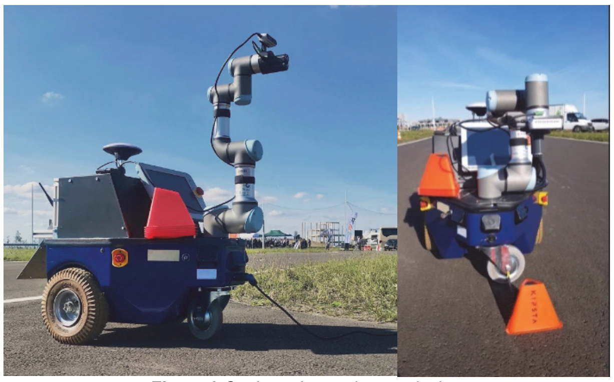


Figure 3 Coning robot and cone placing

The robot manipulator is a small robotic arm (Universal Robots UR3, the smallest member of the Universal Robots family), enhanced with a unique, self-designed 3D-printed gripper.

The role of sensors is very important. The GNSS sensor measures the geographical coordinates, and the camera image sensor detects the environment. RTK refines the data obtained to centimetre accuracy. The function of the LiDAR sensor is laser-based remote sensing (navigation points, obstacles). Together with these, the position of the robot can be identified very accurately on the map, and obstacles can be avoided (see also the next subsection).

Although, as mentioned, our thesis does not aim at a detailed analysis of the physical structure of the robot, the main components are also presented in a technical summary in the appendix.

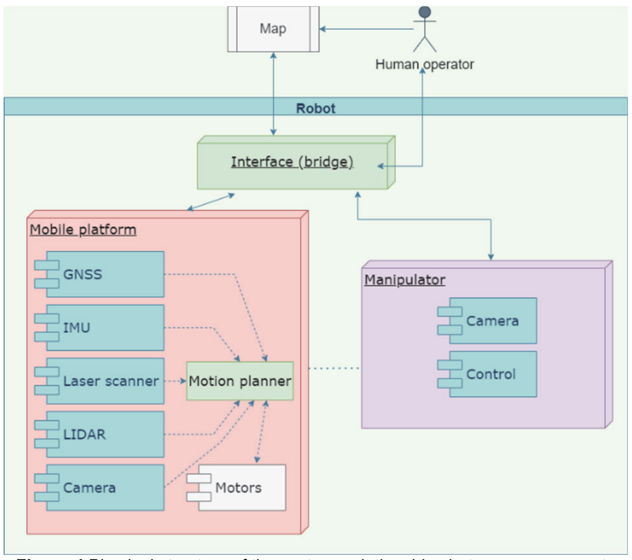


Figure 4 Physical structure of the system: relationships between components

3.2 Logical and Software Architecture

Considering the software part, this robot is a rather complex structure consisting of distributed systems.

According to the logic of operation, several levels can be distinguished, of which the lower level of hardware (control of safety-critical tasks, sensors and motors) is not covered in detail in this paper. At the top level, run the tasks that are now primarily important to us (object detection, trajectory planning). Here, the higher computing requirements and high memory requirements are paramount (as opposed to completely safety-critical operation), that is why embedded (powerful) NVIDIA Jetson AGX Xavier computers (specifically for developing AI applications) are equipped here.

Note that the operating systems running on the computing units of the mobile robot platform are all Linux based, and the development is open source by default.

In order to ensure the connection between the mobile robot platform and the robot manipulator, it became necessary to create a distributed communication network. For this, the ROS 2 (Robot Operating System 2) foxy environment was used (to develop robot applications) [16]. Even within subsystems, each device communicates with other devices in this way (sensors, LiDAR, etc.).

A very positive feature of the ROS system is its ease of development (compared to other ecosystems). In this project, we chose ROS 2 specifically because ROS 1 has a master that needs to be run (similar to the client-server architecture in this), while in ROS 2 everyone is at the same level, so everyone can directly read the other's messages (broadcast technology), and for us the latter was more advantageous here.

Finally, we mention that ROS 2 is based on the DDS (Data Distribution Service) standard, which definitely supports the integrated secure operation of distributed systems.

3.3 Localization and Environmental Representation

If an autonomous device is able to move, it is obviously necessary for it to be aware of its exact physical/spatial position and, if applicable, to be able to identify its position on a map of the physical environment.

We deal with these activities in the next two subsections.

3.3.1 Localization

Very accurate position information is required to perform the sequence of actions required for the robot (see above). The process of gathering this information is localization, which is thus obviously an extremely important and critical activity. For localization, odometry estimation (based on motor encoder) and GNNS sensors with RTK refinement are used as usual.

The odometry estimate is based on the fact that the current physical position and direction angle can be calculated from the current position/speed of the wheel and the axle (and from previous movements, forward travel, turns, etc.) based on the circumference of the wheel—roughly; in an ideal situation exactly.

Table 1 Odometry

Components	Formulas
$\theta = \dfrac{v_{\text{right}} - v_{\text{left}}}{d_{\text{wheels}}}$	$\text{angular}_z = \sum \theta$
$dx = \dfrac{v_{\text{right}} + v_{\text{left}}}{2} \cos\theta$	$\text{linear}_x = \sum dx$
$dy = \dfrac{v_{\text{right}} + v_{\text{left}}}{2} \sin\theta$	$\text{linear}_y = \sum dy$

There are, naturally, other, more accurate ways to determine the robot's current direction angle (orientation). Our classic solution for this was to move the robot (so the direction was measured from the GPS-UTM data). Our plan is to install the two GPS sensors on the robot in the future, and the exact orientation of these can be determined without moving them.

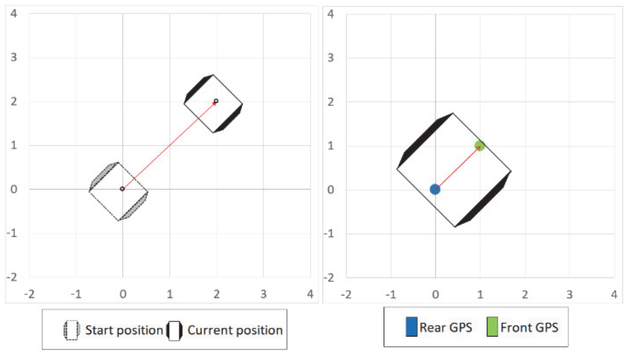


Figure 5 Determination of direction angle using movement, or two sensors, respectively

Closely related to localization is the task of motion and trajectory planning, which, however, already presupposes knowledge of the environment (identification can be performed with the knowledge of the map see environmental representation in the following subsection).

The robot uses cameras and LiDAR to detect cones and obstacles. Knowing of the edge of the pavement is crucial, we know this from the map information, but for security reasons we also look at it with the camera. In addition, moving obstacles on the track (e.g. people on the track) may be detected by the LiDAR sensor.

3.3.2 Environmental Representation

The measured/given data of the physical environment, obstacles and targets, together with the map-environment information, are organized in an environmental representation, which is used as a decision space by motion and trajectory planning algorithms.

Creating the right environmental representation is therefore an extremely important part of the project. During the design, following the ROS REP 105 recommendation, we created a framework for this, moving from the macro-environment to the micro-environment. These are:

- World or map frame: available map and geographic information (database).
- Global framework: terrain (with obstacles) surveyed and adjusted by the operator, combined with map information.
- Local frame: the characteristics currently measured by the robot from the near environment, united with the corresponding characteristics of the global frame.

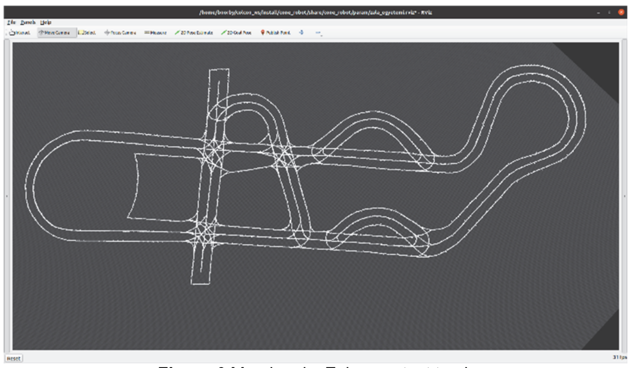


Figure 6 Map in rviz, Zalazone test track

The general maps actually available (based on OpenStreetMap) were not accurate enough for us. Therefore, after proper analysis, the solution was to create a map from our own point cloud in the rviz application (3d visualization tool in ROS). It runs on a separate computer and the operator can select the positions of the cones on it (ROS 2 communication with the robot).

4 TRAJECTORY DESIGN

Considering the robot and its components, we have to separate the task of motion and trajectory planning.

Motion planning is logically close to hardware.

Platform motion planning essentially means activating motors and asking for their states (essentially: local motion). All of this has been discussed above, and we will not discuss this in more detail in the present work. The motion planning of the robotic arm is also reviewed only in summary. Here, for us, it is enough for the platform to deliver the information to the arm in the right position (cone on: pack, successful return communication: I'm done).

The trajectory or route planning deals with finding the optimal trajectory of the robot (this applies to the platform only). It is immediately perceivable that this is a rather complex problem, as the robot (navigation system) has to find the shortest (fastest) route on the map on the one hand, and on the other hand to detect possible static or dynamic changes (and must respond to them).

In the next subsection, we generally present our analysis related to trajectory planning and the algorithms applied.

4.1 Analysis, Algorithms

In the design, we considered the above framework representation as the basis, and in the analysis we came to the conclusion that the currently existing (known) motion planning algorithms (and architectures) provide different performance in each framework. A general, ubiquitous ideal solving algorithm is not available. Therefore, a proprietary motion planning framework has been developed, which includes existing algorithms with modifications or expansions, respectively.

At the map or global framework level, the most important is that the planned path should be (predictably) barrier-free and as short as physically possible (a "compromise on time performance"). Selected algorithms: Dijkstra, *A**, resp. Hybrid *A** and DWA.

At the local frame level, good performance over time is very important, but spatial optimization of the route is not primary and often cannot be guaranteed (due to bypasses). Selected algorithms: Elastic band.

Based on these, the steps of trajectory planning according to our model and plan are as follows (see Fig. 7):
1. Aligns the cone locations with the Dijkstra (greedy) algorithm relative to the robot's current position.
2. The system designs the basic trajectory.
a. using *A** (this was actually used for analysis only),
b. hybrid *A**,
c. dynamic Window Approach (DWA) algorithm.
3. The system refines the trajectory with the Elastic Band algorithm.

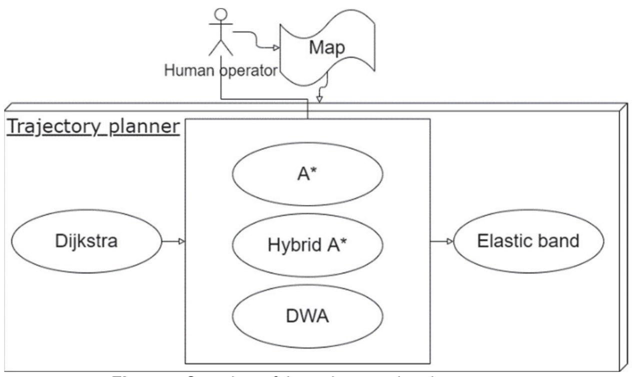


Figure 7 Overview of the trajectory planning process

The *A** algorithm ([17], presented first in 1968) is an extension of the greedy Dijkstra method ([18], 1959) presented earlier, an optimal path-finding algorithm that also uses a heuristic.

Hybrid *A** is one of many enhancements to the *A** algorithm. Here, the transitions of the state space are continuous, not discrete. In fact, this condition cannot be fulfilled perfectly because map representation will always be discrete in practice. On the other hand, a good approximation of the continuous state space allows a much finer representation, so that the advantageous properties of the algorithm are mostly preserved ([19], 2010). Note that temporal performance deteriorates due to the above.

The Dynamic Window Approach (DWA) is a strategic algorithm developed for collision avoidance that has been developed specifically to take into account the dynamic properties of moving robots. Instead of traversing the whole space, it always focuses on only one part of the window space (frame) (it looks forward a few steps) and produces the optimal solution in it ([20], 1997).

Using the results of other route planners, the Elastic band algorithm skilfully refines the route (local improvement of an existing global route) using "bubbles", in a collision-free manner, even taking into account moving obstacles. The procedure uses two main steps: a band refinement step (including the removal of unnecessary bubbles, the correction of path errors) and an optimization step ([5], 1993).

Note that for all algorithms, several variants developed from the basic procedure are common.

These algorithms have been known for some time and, as it was mentioned in the introduction, have been used by many in similar projects to manage autonomous devices.

The novelties of our project are available in design and implementation.

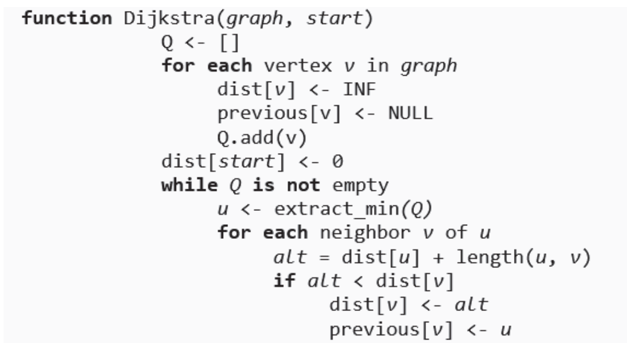
The whole implementation is according to a pure ROS 2 ecosystem. In the main steps of trajectory design, the algorithms build on each other (Dijkstra-Hybrid *A**-Elastic band). The entire software is modular in structure, with components interchangeable in the appropriate module (for example, pulling out *A** and replacing it by Hybrid *A** or DWA). The applied version of each algorithm is (naturally) our own implementation (Python environment, open source) and is improved with own ideas.

4.2 Detailed Description of the Implementation

To design the basic trajectory, we use a very simple, binary matrix-based map representation, which contains only 0 (free) or 1 (fixed obstacle) symbol, respectively.

In the first step, we lined up the cone locations with the Dijkstra algorithm. Its Python-like pseudocode can be seen in Fig. 8.

As is well known, the efficiency of the algorithm is $O(|E| + |V|^2) = O(|V|^2)$, where E is the number of edges and V is the number of vertices. So this is a good polynomial procedure [21].



```
function Dijkstra(graph, start)
          Q <- []
          for each vertex v in graph
               dist[v] <- INF
               previous[v] <- NULL
               Q.add(v)
          dist[start] <- 0
          while Q is not empty
               u <- extract_min(Q)
               for each neighbor v of u
                    alt = dist[u] + length(u, v)
                    if alt < dist[v]
                         dist[v] <- alt
                         previous[v] <- u
```

Figure 8 Dijkstra's algorithm

4.2.1 A* and Hybrid A* Algorithms

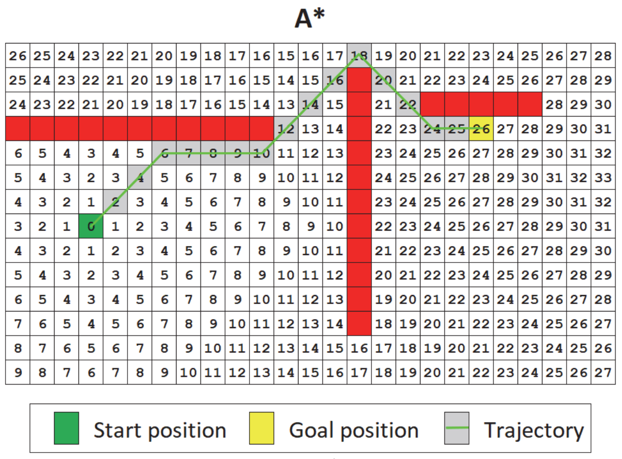
The basic A^* algorithm uses a cost function $f(n) = g(n) + h(n)$ that contains a current cost as the first term (until the given n-th point is reached) and a heuristic estimate to reach the last point from the n-th point.

The procedure is theoretically proven to be optimal in the case of a discrete state space, but when examined under our circumstances, it will not, for the reason that it only passes through adjacent grid points and thus allows only 45-degree turns (so it does not allow really fine tracking). With a very fine grid, naturally, we would get closer to the optimum.

In addition, it is also a negative circumstance that the algorithm requires a lot of memory, it stores all traversed paths (with logical step-backs) and nodes.

The temporal complexity of the algorithm can be exponential depending on the state space and the heuristic, $O(b^d)$, where d is the length of the shortest path and b is the branching factor. However, if the state space is tree-structured, the target is a clear state, and a reasonable constraint is met on our heuristic, the algorithm will be "friendly" polynomial at runtime [6, 22].

The Hybrid A^* algorithm is not limited to grid points due to the assumption (approximation) of a continuous state space. Here we can enter the angle of rotation of the robot (steering wheel). By specifying a larger turning angle (e.g. 30 degrees), a shorter running time can be expected, but the planned route will be more inaccurate. At a smaller angle (e.g. 10 degrees) the running time will be longer, but the planned route will be closer to the optimal one.



26	25	24	23	22	21	20	19	18	17	16	15	16	17	18	19	20	21	22	23	24	25	26	27	28
25	24	23	22	21	20	19	18	17	16	15	14	15	16	🟥	20	21	22	23	24	25	26	27	28	29
24	23	22	21	20	19	18	17	16	15	14	13	14	15	🟥	21	22	🟥	🟥	🟥	🟥	🟥	28	29	30
🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	12	13	14	🟥	22	23	24	25	26	27	28	29	30	31
6	5	4	3	4	5	6	7	8	9	10	11	12	13	🟥	23	24	25	26	27	28	29	30	31	32
5	4	3	2	3	4	5	6	7	8	9	10	11	12	🟥	24	25	26	27	28	29	30	31	32	33
4	3	2	1	2	3	4	5	6	7	8	9	10	11	🟥	23	24	25	26	27	28	29	30	31	32
3	2	1	0	1	2	3	4	5	6	7	8	9	10	🟥	22	23	24	25	26	27	28	29	30	31
4	3	2	1	2	3	4	5	6	7	8	9	10	11	🟥	21	22	23	24	25	26	27	28	29	30
5	4	3	2	3	4	5	6	7	8	9	10	11	12	🟥	20	21	22	23	24	25	26	27	28	29
6	5	4	3	4	5	6	7	8	9	10	11	12	13	🟥	19	20	21	22	23	24	25	26	27	28
7	6	5	4	5	6	7	8	9	10	11	12	13	14	🟥	18	19	20	21	22	23	24	25	26	27
8	7	6	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
9	8	7	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

🟩 Start position 🟨 Goal position ▨ Trajectory

Figure 9 Trajectory design with the *A** algorithm with extension steps

The cost function of the algorithm changes compared to the base version because turning is more expensive than going straight. Formula: $f(n) = g'(n) + h(n)$, where $g'(n)$ is derived from the original $g(n)$ (case of A^*) by incorporating the turn(s) plus costs.

Nevertheless, the memory requirements and execution costs of the algorithm are very similar to those of A^*.

Fig. 11 illustrates that by entering a smaller steering angle, we are indeed much closer to the optimal route. Given a very small turning angle, due to the finite representation of the space, we would in principle get the optimum, only the runtime of the algorithm would "fly away".

```
function Hybrid_A_Star(start, goal, robot_len, steering, step)
    curr_wp = start
    steering_cost = [1, 0, 1]
    step_cost = [1, 0]
    while curr_wp <> goal
    for i = 1, 3  //steering
        for j = 1, 2  //step
            g = curr_wp.cost-heuristic(curr_wp, goal)
            neighbor.x = curr_wp.x + step[j] * cos(curr_wp.angle)
            neighbor.y = curr_wp.x + step[j] * sin(curr_wp.angle)
            neighbor.angle = curr_wp.angle + step[j] *
                    tan(rad(steering[i])/robot_len)
            if neighbor not in obstacle
                neighbor.h = heuristic(neighbor, goal)
                neighbor.g = g + steering_cost[i] + step_cost[j]
                neighbor.f = neighbor.h + neighbor.g
                alternativ_trajectory.add(neighbor)
     curr_wp = (min_cost(alternativ_trajectory))
     trajectory.add(curr_wp)
```

Figure 10 Pseudocode of the Hybrid A* algorithm

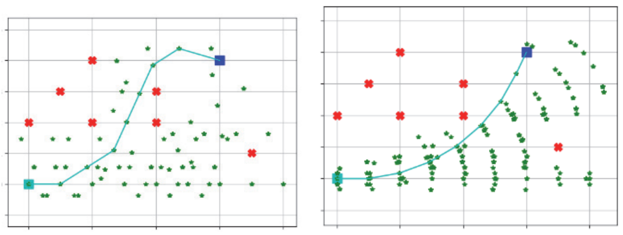


Figure 11 Trajectory design with Hybrid A* algorithm (30 degree or 10 degree steering angle)

4.2.2 DWA and Elastic Band Algorithms

According to the version of the Dynamic Window Approach (DWA) algorithm we have implemented, the robot does not make a fixed turn, but can rotate in an angular range. The navigation looks a few steps ahead (based on a given number limit) and thus builds alternative routes from that point. Of these, it always selects the lowest total cost. In reality, however, at the end of each analysis phase, we take only one step forward and then start counting the cost of the remaining trip again.

Increasing the number limit (looking at more progress) will bring the route closer to optimal, but it will also take longer to run (planning time).

The cost function of the procedure is as follows (taking into account the current position and the target position):

$$c(n) = \min \sum_{step=1}^{n} dist\left(pos_{act}, pos_{term}\right) \tag{1}$$

With the limitations we use, the algorithm has a polynomial runtime, and the memory required is obviously less than that of the A^* or Hybrid A^* method. An additional advantage is that this method allows us to re-plan the route faster during runtime, if necessary, than with the other two algorithms (avoiding unforeseen obstacles that appear during movement).

The elastic band algorithm was used in this project only as a last step, for the local optimization of a route already designed by another algorithm.

The main argument of the procedure is the trajectory (to be improved) already created with the other route planner and for this stretches the new, improved route as a flexible thread to it or for parts of it, and then applies a continuous approach to it.

```
function DynamicWindowApproach(start, goal, goal_radius,
        min_omega, max_omega, min_step, max_step)
    trajectory.add(start)
    wp=start
    while distance(wp, goal) >= goal_radius:
        for i, angle in linspace(min_omega, max_omega)
            omega.add(angle)
            cost[i]=0
            for step in linspace(min_step, max_step)
                curr_wp.x = wp.x + step* math.cos(omega)
                curr_wp.y = wp.y  + step * math.sin(omega)
                cost[i] += heuristic(curr_wp, goal)
        min_index = index(min(cost))
        new_wp.x = wp.x + math.cos(omega)
        new_wp.y = wp.y  + math.sin(omega)
        trajectory.add(new_wp)
    trajectory.add(goal)
```

Figure 12 Pseudocode of the DWA algorithm (version used)

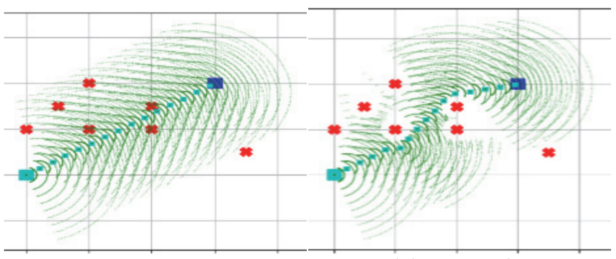


Figure 13 Trajectory design with DWA algorithm (70° steering, 3 steps, part a: obstacle = point; part b: obstacle = circle)

In addition, the procedure shall specify the minimum rate of correction between adjacent steps (tiny constant value); if we can no longer overcome this, the process will stop. For security reasons, the maximum number of steps is also built into the program. The fourth argument of our procedure (small distance value) determines how close the robot can approach obstacles.

Instead of the original bubble approach, we used the simple sliding average method to continuously improve the route. Despite the simplicity, we value this as a significant innovation because we have never seen such an implementation.

The cost function of the algorithm is more difficult to grasp here, as it depends on the "thoroughness" of the review. However, due to local constraints and obstacles (number of points and repair rate), the procedure only works with a relatively small part of the total space, for a very limited time, so also will have a polynomial runtime.

```
function ElasticBand(trajectory, zoom_count, zoom, obstacle_size)
    i = 1
    curr_zoom = 1
    while (i <= zoom_count) and (curr_zoom >= zoom)
        old_trajectory = trajectory
        trajectory = near_trajectory(old_trajectory, obstacle_size)
        curr_zoom=proportion(trajectory, old_trajectory)
        i++

function near_trajectory(old_trajectory, obstacle_size)
    for curr, old_wp in old_trajectory
        new_wp[curr] = average(wp[curr-1], wp[curr], wp[curr+1])
            if distance(obstacle, new_wp) > obstacle_size
                trajectory.add(new_wp)
            else
                trajectory.add(old_wp)
    return trajectory
```

Figure 14 Pseudocode of the elastic band algorithm (version used)

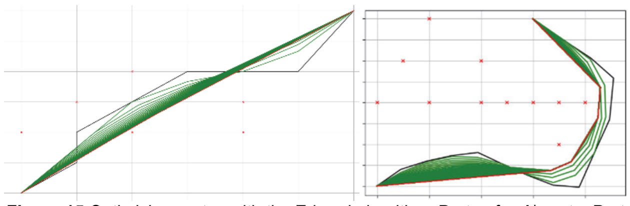


Figure 15 Optimizing routes with the E-band algorithm; Part a: for *A** route; Part b: DWA route; the (red) dots are the obstacles

4.2.3 Gazebo Simulation

The robot is a very valuable device, it is absolutely necessary to protect it from possible damage and collisions. Therefore, before real, physical testing, it is advisable to check the implemented algorithms in a "non-live" environment with simulation. For this, we chose the Gazebo environment, where we built an exact copy of our robot as a digital twin the various parameters of the robot can be specified in a suitable file (unified robot description format; urdf).

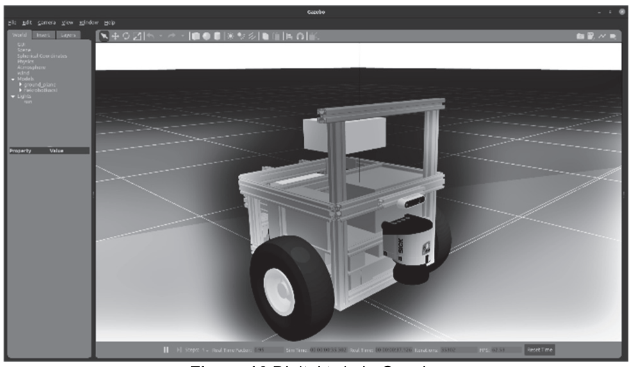


Figure 16 Digital twin in Gazebo

5 SUMMARY, FURTHER PLANS

We feel that the goals set out in advance have been adequately met in this project. It is especially noteworthy that this was a real practical engineering job in which the components of the robot were essentially built from zero. We also consider the ROS 2 communication between the platform and the robotic arm to be extremely valuable, and their modular architecture, respectively.

Turning to the main part of our current work, it should be emphasized that the same modular structure was achieved in trajectory design (interchangeable algorithms). It is also interesting and valuable that the implemented algorithms can be built on top of each other. We consider the improvements of the algorithms we have implemented to be novel and useful. We have used several currently very modern solutions (e.g. modern environments, digital twin concept).

Our general goals for possible improvements include increasing accuracy (motion planning); possibility to pack large cones; collecting cones (and other items); improving security; designing a portable own base station for RTK; implementation of a robot-drone connection; 5G communication.

Due to the size of the finished prototype and the parameters of its gripper, it is only suitable for laying small sports cones. It can be used on various sports fields and obstacle courses. A new, larger 1 m × 1 m based platform is being built, which is already suitable for laying large traffic cones and serving the test track in the automotive industry.

We will replace the robot controller with a Pixhawk drone controller. This controller better facilitates the processing of sensory and motor data. We will also use Pixhawk integrated into the ROS 2 ecosystem. This controller will also help our further plan to build a platform-independent ecosystem.

The collection of cones is underway in the area designated by the operator. Our long-term goal is that not only the buoys will be collected by the robot, but also any objects that do not fit on the test track and are left there (rubbish, abandoned bags, coats, etc.).

Our even longer-term goal is for a drone to first map the test track for mismatched devices, and then the drone sends the robot out to collect them instead of the human operator.

Acknowledgements

The research was supported by the Ministry of Innovation and Technology and the National Office for Research, Development and Innovation (Hungary) within the framework of the National Laboratory Program for Autonomous Systems.

6 REFERENCES

[1] Siciliano, B. & Khatib, O. (2016). *Springer Handbook of Robotics*. Berlin-Heidelberg: Springer Verlag. https://doi.org/10.1007/978-3-319-32552-1

[2] Kanade, T., Thorpe, C., & Whittaker W. (1986). Autonomous land vehicle project at CMU. *Proceedings of the 1986 ACM fourteenth annual conference on Computer Science-CSC '86*, 71-80. https://doi.org/10.1145/324634.325197

[3] Hamid, U. Z., Pushkin, K., Zamzuri, H., Gueraiche, D., & Rahman, M. A. (2016). Current collision mitigation technologies for advanced driver assistance systems-a survey. *PERINTIS e Journal*, *6*(2), 78-90.

[4] Kurzer, K. (2016). *Path Planning in Unstructured Environments: A Real-time Hybrid A* Implementation for Fast and Deterministic Path Generation for the KTH Research Concept Vehicle*. MSc Thesis, Stockholm: KTH Institute of Technology.

[5] Quinlan, S. & Khatib, O. (1993). Elastic bands: connecting path planning and control. *Proceedings IEEE International Conference on Robotics and Automation*, *2*, 802-807. https://doi.org/10.1109/ROBOT.1993.291936

[6] Zeng, W. & Church, R. (2009). Finding shortest paths on real road networks: the case for A*. *International Journal of Geographical Information Science*, *23*(4). https://doi.org/10.1080/13658810801949850

[7] Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., & Jurišica, L. (2014). Path Planning with Modified A-Star Algorithm for a Mobile Robot. *Procedia Engineering*, *96*, 59-69. https://doi.org/10.1016/j.proeng.2014.12.098

[8] Sánchez-Ibáñez, J. R., Pérez-del-Pulgar, C. J., & García-Cerezo, A. (2021). Path Planning for Autonomous Mobile Robots: A Review. *Sensors*, *21*(23). https://doi.org/10.3390/s21237898

[9] Fragapane, G., de Koster, R., Sgarbossa, F., & Strandhagen, J. O. (2021). Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. *European Journal of Operational Research*, *294*(2), 405-426. https://doi.org/10.1016/j.ejor.2021.01.019

[10] Cebollada, S., Payá, L., Flores, M., Peidró, A., & Reinoso, O. (2021). A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data. *Expert Systems with Applications*, *167*. https://doi.org/10.1016/j.eswa.2020.114195

[11] Simon, J., Trojanova, M., Hosovsky, A., & Sarosi, J. (2021). Neural network driven automated guided vehicle platform development for industry 4.0 environment. *Tehnički Vjesnik-Technical Gazette*, *28*(6), https://doi.org/10.17559/TV-20200727095821

[12] Zheng, L. (2022). Predictive control of the mobile robot under the deep long-short term memory neural network model. *Computational Intelligence and Neuroscience*, *2022*. https://doi.org/10.1155/2022/1835798.

[13] Zhang, C., Jiang, X., Liu, X., & Zhang, D. (2021). Single-Loop full R joints of multi-mode omnidirectional ground mobile robot. *Tehnički Vjesnik-Technical Gazette*, *28*(6). https://doi.org/10.17559/TV-20200804125931

[14] Hartzer, J. & Saripalli, S. (2021). AutoCone: an omnidirectional robot for lane-level cone placement. *Arxiv.* https://doi.org/10.1109/IV47402.2020.9304683

[15] Hajdu, C., Boros, N., & Ballagi, Á. (2021). Development of track-building mobile robot based on cognitive-inspired motion planning architecture. *Proc. 12th IEEE International Conference on Cognitive Infocommunications Cog Info Com*, 781-786.

[16] Quigley, M., Conley, K., Gerkey, B. P., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). ROS: an open-source robot operating system. *ICRA Workshop on Open Source Software*, *3*(3.2).

[17] Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, *SSC-4*/2, 100-107. https://doi.org/10.1109/TSSC.1968.300136

[18] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269-271. https://doi.org/10.1007/BF01386390

[19] Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Practical search techniques in path planning for autonomous driving, *The International Journal of Robotics Research*, *29*, 485-501. https://doi.org/10.1177/0278364909359210

[20] Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23-33. https://doi.org/10.1109/100.580977

[21] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. (3rd ed.), MIT Press and McGraw-Hill.

[22] Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Boston: Addison-Wesley.

Contact information:

Norbert BOROS, assistant research fellow, council administrator
(Corresponding author)
SzéchenyiIstván University, Győr,
H-9026, Győr, Egyetemtér 1., Hungary
E-mail: bnorby@sze.hu

Gábor KALLÓS, associate professor
SzéchenyiIstván University, Győr
H-9026, Győr, Egyetemtér 1., Hungary
E-mail: kallos@sze.hu

Áron BALLAGI, associate professor, head of Autonomous and Intelligent Robotics Laboratory, head of Department of Automation
SzéchenyiIstván University, Győr
H-9026, Győr, Egyetemtér 1., Hungary
E-mail: ballagi@sze.hu

Table 1 Components of the robot

Device name	Device photo	Decision consideration (comparison)
Neobotix Mobile Robot MP-500 robotplatform		A simple robot platform, easily expandable, it is important for us mainly because of the drive
Universal Robot UR3 collaborative robot arm		Due to its 6-degree-of-turn design, it is capable of skilful/versatile movement
Pixhawk 4 controller (PX4 Autopilot)		We chose it because of the PX4 firmware, it provides many useful services, the sensor data comes filtered at the basic level
Drotek F9P RTK rover		Compatible with many devices, reliable and easy to-handle
Drotek DA910 multi-band GNSS Antenna		External antenna for the previous device, used to increase accuracy
Sick laser scanner		Reliable and easy to-handle