Virtual testing ground for the development of control systems for unmanned vehicles in agriculture

Kirill Svyatov\textsuperscript{1}, Ivan Rubtcov\textsuperscript{1}, and Aleksey Ponomarev\textsuperscript{1}

\textsuperscript{1}Ulyanovsk State Technical University, 432027, Severny Venetc, 32, Ulyanovsk, Russia

\textbf{Abstract.} The article describes the technology for deploying and programmatically setting up a virtual test environment for testing algorithms for ground-based agricultural unmanned vehicles. The fields2cover library is used to plan the path, and the Webots simulator integrated with the robot operating system is used as the basis for the test suite. The main task to be solved is to match the coordinate system of the path through the field and the coordinate system of the virtual autonomous vehicle. It also describes the process of creating digital maps using OpenStreetMap, converting them into the necessary formats for generating a route in the fields2cover library, and setting up a simulator based on Webots. The work is relevant for educational purposes and is of interest to developers involved in the design of autonomous vehicles.

\section{Introduction}

Currently, the market for unmanned vehicles (UV) is growing rapidly \cite{1} and may reach $1,600$ billion by 2030 and $7$ trillion by 2050 \cite{2}. Among all types of unmanned vehicles, the important are agricultural ground autonomous vehicles, which allow solving several important problems \cite{3}:

\begin{itemize}
  \item Increasing yields through the use of precision farming technologies through the use of global positioning systems and precise field cultivation.
  \item Exclusion of the factor of harmful production for machine operators cultivation fields with chemicals.
  \item More efficient use of machinery by fewer operators.
  \item Generation of multi-layered digital field maps for further application of analytical tools for precision farming: yield maps, fertilizer application maps (pesticides, herbicides), crop rotation maps.
  \item Automatic route planning and traffic optimization reduce unnecessary trips and optimize fuel or electricity consumption. This leads to a more efficient use of resources and a reduction in the ecological footprint of agricultural activities.
\end{itemize}

* Corresponding author: k.svyatov@ulstu.ru.
The development of a control system for UV is driven by four basic processes: localization, recognition, planning and control [4].

Localization provides estimation of the position of the UV in global coordinates, as well as relative to obstacles or objects that are important for solving the applied problem. The recognition module analyzes the environment with sensor data processing algorithms. The main sensors for determining the parameters of the external environment used in UV for agricultural purposes are lidar, photo camera, stereo camera, global positioning system (GPS/Glonass, inertial system, compass). The planning module is designed to compute a UV route. It uses localization data and information about environmental objects obtained at the previous stages of data processing, as well as maps of several types:

- Global map (OpenStreetMap, Yandex, etc.) showing publicly available information about field boundaries, roads, pillars and the nature of the landscape.
- Local map of static obstacles - is generated when following the planned path and is updated as it passes again, is saved for permanent use, because static obstacles rarely change.
- Local map of dynamic objects with key characteristics - is generated only when following the path and is limited by the field of view of the sensor components.
- Local map of the application task execution – contains a list of coordinates of objects important for the application task execution. For example, the beginning of the route, the yield of the plot, the area for loading and unloading, the area for canceling processing, etc.

The control components contain structural mechanisms, electronic components and software modules for controlling the vehicle to ensure that the generated route is followed.

The process of developing control algorithms for UV involves constant testing. Testing on a real car is expensive and difficult. Therefore, simulators are now popular, which make it possible to simplify the process of developing and debugging algorithms and models for unmanned driving. The use of simulators allows solving the problems of developing modules in such a way that the developed algorithms can be transferred to a real car with no changes, other modules will require additional configuration and programming during transfer.

Currently, there are no specialized UV simulators for agriculture. A sufficient number of simulators are available in the field of UVs, such as CARLA, NVIDIA Drive Sim, Vista, Roblox, simulators from Matlab and Ansys, but all of them are designed to simulate UV in an urban environment [5]. Agricultural simulators should have a number of special features:

- Agriculture field maps contours;
- Special agents must be present on the scene: domestic and wild animals, agricultural machinery.
- The scene must contain the features of the field relief - ravines, special roads.
- The scene should provide functions for modeling various processes in agriculture, available for automation by means of UV: spraying, spreading fertilizers, plowing, sowing.

In addition, the creation of scenes should be a relatively simple process [6]. To control the UV, it is reasonable to use ROS - Robotic Operation System. The Webots simulator satisfies these requirements.

The purpose of this work is to increase the efficiency of designing control systems for UV in agriculture by creating digital field maps [7], using technologies for building digital field maps, using algorithms for building paths and control systems to follow this path.

To achieve this goal, the following tasks were solved:

- Reverse engineering of the Fields2Cover library [8], intended for planning the route through the agricultural field of the UV.
- A digital map of the field obtained from the OpenStreetMap mapping service was built to integrate with Webors and ROS.
- Implemented mapping of the coordinate system of the scene Webots [9] with the coordinate system of the path through the field obtained from fields2cover.
The algorithm for passing through points generated in the fields2cover library has been implemented.

2 Creating a digital agriculture field model

To create a digital field map, the fields2cover library is used. This library is designed to solve the field coverage path planning (CPP) problem in several ways.

At the first stage, a polygon is created at the given coordinates, which bounds the field. Further, based on the width of the UV, the headland around the field is calculated [8].

The turning area must be reserved for the vehicle. The simplest approach is to allocate an area of constant width around the field. This strategy allocates a large amount of space for an area with low yields. Depending on how the lanes are laid out, some portions of the headland are parallel to the lanes and therefore not needed for the turn. In 2D flat fields, a reference line can be used as a guide to create lanes, where each parallel creates a lane. This line can be chosen for convenience or by an algorithm such as brute force or metaheuristics. Fields2Cover is the only software solution that provides path generation algorithms for ground agricultural robots, including optimizers and objective functions for best path creation, headland support and turn planning.

Next are the tracks. Track generators plan their placement on the field in order to be able to cover it completely, taking into account the width of the working body (sprayer, spreader). In order for the generator to work correctly, it is necessary to determine the objective function [10], on the basis of which they will search for the best angle. For example, if the objective function is the number of stripes, then using a brute force algorithm, all possible angles will be enumerated and the one that gives the least number of stripes will be left. It is also possible to construct a travel path based on the "sum of lane lengths" criterion. Next, a detour route of the field is planned. For this, several patterns are used. The boustrophedon pattern is one of the most famous patterns for covering the field. The lanes are passed in the simplest order: first the first lane is passed, then the second and so on. This template can have 4 results per field, depending on the starting point. The serpentine order covers the field, skipping one lane each turn and then returning using the uncovered lanes. This pattern, compared to boustrophedon, reduces the number of sharp turns.

Fig. 1. Calculation of turns using Reeds-Shepp curves method

After that, you need to generate the final path. To do this, it is necessary to generate trajectories of turns. To do this, the library provides several options:

- Dubins curves are generated with 3 turn segments. Turn segments in Dubins always go forward. Segment types: straight, right curve or left curve. Dubins Curves allows you to generate the shortest path for a turn. While Dubins curves create the shortest path possible, moving from one curve segment to another results in an instantaneous change in curvature.
The vehicle cannot follow the path with this problem, so this path planner implements an integrator to smoothly change the curvature.

- Reeds-Schepp curves also compute the shortest path but allow the vehicle to move backwards (Fig. 1).

There are web applications such as GAOS that allow farmers to design or customize coverage paths through a user-friendly interface. Many of these web applications, despite being of great help to the farming community, have been developed in collaboration with companies, limiting the ability to make the code publicly available.

Although seven other designs have been found, none of them can be adapted for agricultural purposes with ground-based robots. Turning maneuvers for ground-based robots must be planned to move from one lane to another. Unfortunately, some packages only compute the route to cover a region. These packages are for quadcopters or indoor robots. However, the code needs to be changed to support path generation for nonholonomic robots. Fields2Cover is the only software solution that provides path generation algorithms for ground agricultural robots, including optimizers and objective functions for best path creation, headland support and turn planning.

3 Design a self-driving car agent

A simulation test suite is a set of related components available in the Webots environment:

- Polygon scene, .wbt file
- An object model of a car added to the scene

A package for ROS2 that defines the following: URDF description file for the BPTS, which describes the elements of the robot, as well as plug-ins for receiving data from the Webots environment from virtual sensors; configuration file for the RVIZ rendering environment; ROS2 launch file that describes how to launch ROS2 nodes and TF2 static transformation schemes; a node where data from sensors is read and dynamic transformations are generated to solve the problem of basic localization, as well as transform data received from sensors.

For UV, you need to add the following sensors:

- Velodyne VLP-16 16-channel lidar
- 640x480 resolution camera
- Stereo camera (Range Finder)
- Compass
- Inertial system
- GNSS system (GPS)
- Accelerometer
- Radar
- Gyroscope

Data received from configured sensors is possible to display in the rviz environment (Fig. 2).

The controller type must be set to "external" in order to be able to be controlled externally via ROS2.

In order to provide localization in the ROS environment, it is necessary to configure the obtaining data from Webots and provide the generation of messages to the topics /gps and /odom.

It is also necessary to provide static transformation of frames:

- map -> odom
- base_link -> lidar
- base_link -> imu_link
To calculate the correspondence of coordinates in the coordinate system of the field and the vehicle, in addition to the indications of latitude and longitude, it is necessary to determine the course of the vehicle. This is done from the readings of the inertial system. But since the angular position data is generated in quaternion format, then the transformation into Euler angles is performed in the following way:

\[
\begin{align*}
\text{roll} & = \arctan\left(\frac{2wx + yz}{1 - 2(x^2 + y^2)}\right) \\
\text{let } p = 2(wy - zx) \text{ then pitch} & = \sin(t), \text{ where } t \\
& \begin{cases} 
1, & \text{if } p > 1 \\
-1, & \text{if } p < -1 \\
p, & \text{if } -1 \leq p \leq 1
\end{cases} \\
\text{yaw} & = \arctan\left(\frac{2wz + xy}{1 - 2(y^2 + z^2)}\right)
\end{align*}
\]

Because navigation is performed in two-dimensional space, then we are only interested in the parameter \(\text{yaw}\), which will be used in further calculations as UV angle.
In general, the implemented data processing flow in the car control system in the simulation environment is as follows (Fig. 3)

4 Running a path in the simulator

To obtain a digital map of a field (Fig. 4), the user needs to determine the GPS coordinates of this field, enter them into the configuration file located at /home/{user_name}/catkin_ws/src/fields2cover_ros/data/example1.xml. After modifying this file and launching the standard fields2cover solution in ROS, it will be possible to display field maps in rviz with marked lanes and turns based on the specified data. You can also change the width of the working body of the UV, which will be taken into account when generating a digital map.

![Fig. 4. Digital map for created map displayed in RVIZ](image)

In order to generate a digital map for another field, you need to get its GPS coordinates. This can be done, for example, using the service https://geotree.ru/. After that, you need to change the file located along the path catkin_ws/src/fields2cover_ros/data/example1.xml and add the coordinates of the points that are the field boundaries inside the <gml:LinearRing> tag. The values come in pairs, the first is the longitude of the point, the second is the latitude. The values for the first point must be duplicated at the end to form closed field boundaries.

To get the coordinates of the points that make up this digital map, you need to create a separate ROS package, and in one of the nodes create a listener for the /field/swaths topic, which will receive messages of the Marker type. One of these messages with a list of generated waypoints is saved to a text file for further use in the Webots simulator.

To create a Webots scene, you need to use an importer that allows you to create a Webots scene based on digital OpenStreetMap (OSM) maps. First, the OSM site exports the map part with the field to an .osm file. It is important that this part contains a field, the route of travel for which was determined in the previous step (Fig. 5).

It is necessary to put the car approximately at the point that corresponds to the beginning of the digital map, along which it is necessary to go around the field.
After that, it is necessary to determine the initial coordinates of the car in the global map by reading them from the odom topic, in this case they are 210.2312027069885 and 77.42130129912289, respectively. But in the file that was obtained earlier, the first coordinates are -69.3696542402323 and 142.373412265843, respectively. In order to be able to use a digital map to bypass the field, it is necessary to proportionally change all points in this file by the value by which the first two points in the file differ from the first two points in the simulation according to formulas (4) and (5).

\[ X_{nM} = X_{nM} + \Delta X \]  \hspace{1cm} (4)
\[ \Delta X = X_{SW} - X_{SM} \]  \hspace{1cm} (5)

In the formulas above, $X_{nM}$ is the nth point in the digital map file, $X_{SW}$ is the X start point of the car when the simulation starts, $X_{SM}$ is the X start point in the file. The same formulas must be used to calculate $Y$.

\[ Y_{nM} = Y_{nM} + \Delta X \]  \hspace{1cm} (6)
\[ YX = Y_{SW} - Y_{SM} \]  \hspace{1cm} (7)

In this case, you need to add 279.60086451093118 to X, and add -64.95211096672011 to Y. It is also necessary to remove from the file all characters that are not related to the numerical representation of coordinates and add the following format: in each line, one number X or Y, starting from X.

After the file has been modified, you need to move it to /home/hiber/repositories/cad-self-driving/simulation/webots_ros2_suv/worlds/ and rename it to ulstu_field_points.txt.

To steer the car in such a way that it reaches the coordinates marked on the digital map, it is necessary to store these coordinates somewhere and control the angle of rotation of the wheels depending on the angle between the direction of the car and the coordinate that needs to be reached.

At the moment of node initialization, arrays will be created, in which coordinates from the ulstu_field_points.txt file will be placed, and variables will be created to track the current angle of the car, the current coordinates of the car. Also added a listener for the publisher /odom, which passes the current coordinates of the car and a variable from which you can get the angle of the direction of the car.

Each time data is received from the /odom publisher, a check is made to see if the next coordinate from the digital map has been reached. To do this, the current coordinates of the car are set and the distance between the current coordinate and the next unreached coordinate from the array of digital map points is searched using the calculate_distance() method. If the
distance between these coordinates is less than a certain value, the point is considered reached, and the index that is used to find the next point is increased by one.

Also in this section of the code, the current angle of the vehicle direction is written to the variable, which is used to calculate the required angle of rotation of the wheels to reach the next point from the digital map.

![Diagram](image)

**Fig. 6.** Finding the angle the car needs to turn to reach the next point

The variable that is used to set the wheel angle is `steering_angle`. The available values for this variable range are from minus one to plus one. Accordingly, if the car goes straight, then `steering_angle` is equal to zero, if to the left, then `steering_angle` is equal to minus one, to the right - `steering_angle` is equal to one. The maximum error we can get is 3.14159 radians, the minimum is -3.14159. According to this logic, the error value must be multiplied by a factor of 0.3183, so that at the maximum deviation there is a maximum angle of rotation of the wheels. After finding the angle of rotation of the wheels (8), a message with this value is sent to the car. The error value is sought as the difference between the angle of the vehicle's direction of travel and the angle between the current point and the next point (9).

\[
SA = \text{error} \times \text{coeff} \tag{8}
\]

\[
\text{error} = \alpha_1 - \alpha_2 \tag{9}
\]

- \( SA \) - steering angle of wheels, \( \text{error} \) - error value, \( \alpha_1 \) - angle of vehicle movement direction, \( \alpha_2 \) - angle between current point and target, \( \text{coeff} \) - proportional coefficient.

In this figure (Fig. 6), the angle between coordinates \( A \) and \( E \) is 0.7853 radians, but since the angle of movement of the vehicle (marked in green) is 1.5708, then the control action directed to the wheels will be \((1.5708 - 0.7853) \times 0.3183 = 0.2500\). The same logic works for other angles as well.

The functions that calculate the distance and angle between points are simply a representation of mathematical functions in code. When you start the car successfully passes the route indicated in the digital map.

**5 Conclusion**

This article describes the process of preparing a simulator for the passage of agricultural unmanned vehicle across the field in terms of preparing the trajectory of movement by external libraries, as well as programming the control system of the UV to ensure travel along the designated trajectory with the development of a module for coordinating the UV
coordinate system in the simulator and in the planned trajectory. Coordination methods, as well as the technological aspect of preparing the simulator, claim novelty. Further development of the described tools implies the connection of an obstacle avoidance module and a dynamic model of the behavioral analysis of the scene. The developed algorithms can be used in the process of teaching students the basics of designing an UATS, as well as for designing an UATS in the planning and motion control modules.

6 Acknowledgments

This work was supported by RSCF Grant 23-11-00265

References

2. Troy Baltic, Alex Cappy, Russell Hensley, Nathan Pfaff. The future of mobility is at our doorstep. Available online: URL https://www.mckinsey.com/-/media/McKinsey/Industries/Automotive%20and%20Assembly/Our%20Insights/The%20future%20of%20mobility%20is%20at%20our%20doorstep/The-future-of-mobility-is-at-our-doorstep.ashx (accessed on 08.06.2023)