An Integration Framework for UGV Outdoor Navigation System Based on LiDAR and Vision Data

Ashraf Saleem, Ahmed Al Maashri, Lazhar Khriji, Medhat Hussein
Department of Electrical & Computer Engineering
Sultan Qaboss University
Al-Khoudh, Sultanate of Oman
{asaleem, amaashari, lazhar, medhatha}@squ.edu.om

***Abstract* — In this paper, we present an architecture to fuse different data from onboard sensors (LiDAR and Camera) for real-time navigation of Unmanned Ground Vehicle (UGV). The proposed system architecture comprises three main interacting modules; line and obstacle detection module, path planning and tracking module, and a real-time motor driving processing module. While these modules were developed individually on different platforms, however, the integration framework that we developed allowed these modules to coexist and interact seamlessly. The UGV prototype was put to the test in a variety of outdoor environments, both cluttered and uncluttered, to navigate safely within limited area bounded by lines. While the work on the proposed architecture is ongoing, this paper shows initial results of system platform development, modules integration framework, and real-time control accuracy and performance.**

Keywords — UGV; Robotics; Data fusion; real-time navigation

I. INTRODUCTION

Unmanned Ground Vehicles (UGVs) are a type of mobile robots, which have numerous applications both in military and civilian domains. For example, in the military domain, UGVs might be used to detect, neutralize, and breach minefields and other obstacles [1]. In contrast, UGVs has the potentials to help in rescuing disaster victims by exploring hazardous environment that are dangerous to human beings [2].

UGVs are controlled either remotely or autonomously. An autonomous UGV requires a whole host of sophisticated algorithms to allow the vehicle to navigate a path and avoid any obstacles. In fact, real-time navigation of autonomous UGVs in outdoor environments is still one of the topics that attract researchers and investigators [3-6].

In this work, we consider a UGV that is travelling outdoor between two points; namely, origin and destination. Here, we assume that the destination is given in the form of Global Positioning System (GPS) waypoint. While travelling, the UGV must remain within a path bounded by two lines and it should avoid obstacles as it encounters them. Additionally, we consider terrain mapping, where the UGV has to build a map of a given area. This map is utilized by path planner and tracker algorithms in order to navigate between obstacles en route to the destination GPS waypoint.

Navigating the UGV in such challenging environments requires developing a control system that would receive multiple sensory inputs. Upon processing these inputs, the controller needs to take a decision that would influence the course of the UGV. Furthermore, it is expected that the UGV is highly responsive to the surrounding environment, which means that the processing and decision making needs to be done in real-time. Given the level of complexity of the scenario presented above, developing a real-time controlling system becomes a challenge.

This paper presents a method where three main processing modules are executed in real-time. These processing modules are: (a) Bounding lines and obstacles detection module, (b) Path planning and tracking module, and (c) Real-time motor driving processing module.

The main contribution of this paper lies in developing a framework that integrates all three modules presented above. Even though these modules were developed separately, however, the proposed framework allows these modules to interact and co-exist without compromising accuracy or speed.

The rest of this paper is outlined as follows: a literature survey of several UGV architectures and control algorithms is presented in Section II. This is followed by a description of the proposed UGV architecture in Section III. Furthermore, Section IV discusses the software framework. Finally, Section V concludes the paper.

II. BACKGROUND

There are a number of efforts for developing UGV architectures. For instance, the architecture proposed in [7] is comprised of command zone electronics and a sensor suite, where the latter incorporates a Light Detection and Ranging (LiDAR) system, vision system, navigation, and an advanced Operator Control Unit (OCU). On the other hand, the UGV system proposed in [8] is designed to operate – in a fully autonomous mode – around an outdoor obstacle course. The entire proposed robotic design is easy to maintain and deploy. Finally, the Nested Hierarchical Controller (NHC) architecture, developed by Meystel [9], represents a typical organization of a mobile robot. It is designed based on sense, plan, and act model.

Several algorithms have been developed to perform autonomous real-time navigation. The renowned Dijkstra's algorithm [10], for instance, finds the shortest path between a

This work was possible through the generous donations of the following sponsors: Sultan Qaboos University, The Research Council, Huawei Technologies Co., Oman Oil Refineries and Petroleum Industries Company, NovAtel Inc.

source and a destination. The downside of the algorithm is that the path calculation is slow. Best-First-Search (BFS) algorithm [11] works similar to Dijkstra's algorithm, however, BFS trades off accuracy for speed. Therefore, while BFS is faster than Dijkstra, however, there is no guarantee that BFS will find the shortest path. To combine the benefits of both Dijkstra and BFS, the A* search algorithm [12] uses two cost functions; namely, the past path-cost and future path-cost. The past path-cost represents the exact cost of the path from the starting point to any vertex. In contrast, the future path-cost represents the heuristic estimated cost from a vertex to the goal. The AD* algorithm [13] – a variant of A* algorithm – is an incremental re-planning algorithm that starts by computing an initial bounded suboptimal solution, then it improves that solution incrementally as time allows. On the other hand, Artificial Potential Field (APF) algorithm [14] is based on Potential Field: the obstacle is represented by high potential and the robot and the destination are represented by low potential. The destination attracts the robot and the obstacle repels the robot. Then, the vector sum of the repulsive and attractive forces are calculated. The drawback of this algorithm is that the robot may get stuck in U-shape and symmetric obstacles. Vector Field Histogram [15] represents the obstacles in a two-dimensional Cartesian histogram grid by a certainty value. Then, a polar histogram with sectors is constructed around the momentary center of the robot. Next, the Cartesian histogram grid is mapped into the polar histogram. Finally, the steering direction is calculated based on the lowest polar obstacle density sectors.

In addition to the algorithms presented above, a number of robot navigation heuristics have been proposed. For example, Parasuraman et al. [16] has devised a fuzzy approach, whereas Maaref et al. **Error! Reference source not found.** has developed a navigation method in an unknown environment based on the combination of elementary behaviors. A new path tracking scheme for a car-like mobile robot based on neural predictive control is proposed in [18].

III. UGV Architecture

This section presents the main components of the developed UGV.

A. An Overview of the UGV Platform

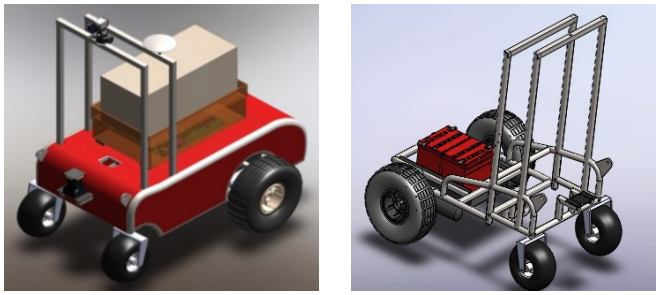
The UGV platform under-test was designed and developed in the College of Engineering, Sultan Qaboos University. The vehicle has four wheels; front caster wheels and motor-driven rear wheels. This UGV supports differential drive. Therefore, each one of the rear wheels is driven by a separate motor.

The UGV platform is 90×60 cm, with a maximum speed of 8 km/hr. The platform is equipped with a mechanical stop that can be used for emergency stop. Fig. 1 shows the CAD model of the UGV platform and the chassis.

The UGV is equipped with four sensory input devices to gather information about the surrounding environment. These sensors are: a LiDAR, an image sensor (i.e. camera), an Inertial Measurement Unit (IMU), and a GPS sensor. The platform hosts a single onboard computer that is used to process all sensory inputs and to generate the necessary commands to the actuators. This onboard computer is a customized computing system with mini-ITX form factor, which will henceforth be referred to as Customized Computing Board (CCB). The CCB platform hosts a 4 GHz quad-core Intel i7 4790K processor, with a total of 8GB RAM.

Fig. 2 illustrates block diagram of the system's main components and the interconnections between all the components. The figure highlights the three modules discussed earlier in the introduction. The components shown in the figure control the behavior of the UGV, and therefore, the UGV architecture should be designed in a way to improve real-time control performance. The following subsections discuss each of these modules in details.

Fig. 3 shows the fully integrated UGV.



(a) (b)

Fig. 1. CAD model of the UGV platform: (a) The UGV platform with a load placed on the platform, (b) CAD model of the chassis showing the location of the battery.

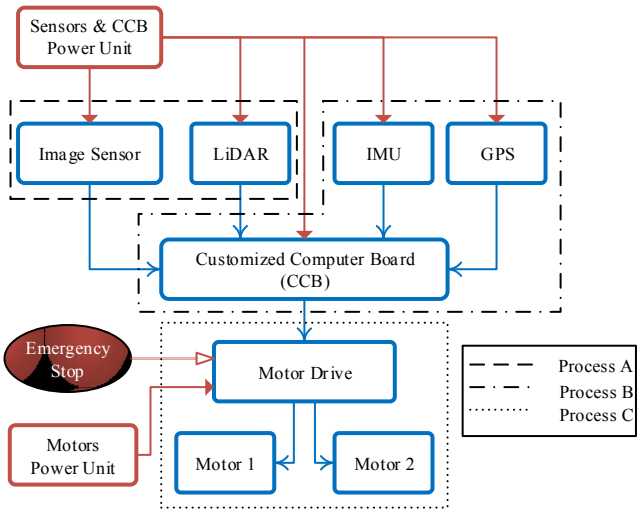


Fig. 2. UGV system components. Note that the system is powered by two distinct power units. The first power unit supplies power to the sensors and onboard computing system, while the second unit is responsible for powering the motors. "Process A" represents the module responsible for detection of bounding lines and obstacles, while "Process B" represents the module responsible for path planning and tracking. Finally, "Process C" represents the module in charge of real-time motor driving.

Fig. 3. The UGV after full integration of all components.

B. Bounding Lines and Obstacle Detection Module

While travelling to the destination, the UGV needs to maintain its course by remaining within the bounding lines and avoiding any obstacles. To do that, the UGV must first detect the lines and obstacles. This subsection discusses the process that we followed to detect the lines and obstacles.

1) Line Detection

An image sensor is used to capture the scene in front of the UGV. The frames acquired from the image sensors are initially in 24-bit RGB format (See Fig. 4.a). This sequence of frames is first smoothed using Gaussian blur filter, which emphasizes the correlation between pixels while reducing the effects of noise.

Then, each frame is converted to HSV color space (See Fig. 4.b) in order to separate the image intensity (*luma*) from the color information (*chroma*). This conversion results in the removal of shadows and retains a frame that is more robust to changes in lighting. Thresholding the frame helps in segmentation of lines, which makes them easier to detect, as shown in Fig. 4.c. This segmentation may result in discontinuities in the lines, which can be resolved using Hough transformation (See Fig. 4.d and Fig. 4.e). The lines detection process was developed entirely using OpenCV library [19].

2) Obstacle Detection

A LiDAR sensor is used to detect obstacles. The sensor has an angle coverage of 270^{o} and can detect objects at distances of up to 30 meters. The sensory data of the LiDAR indicate the heading (i.e. angle) of the obstacle and how far is the object from the UGV (i.e. range). These data are converted into a binary map, which specifies the position of the obstacles using the UGV as a reference. A LabVIEW code was developed to acquire and process data from the LiDAR.

Data fusion is required to combine the sensory inputs from the image sensor and the LiDAR. The data fusion process that

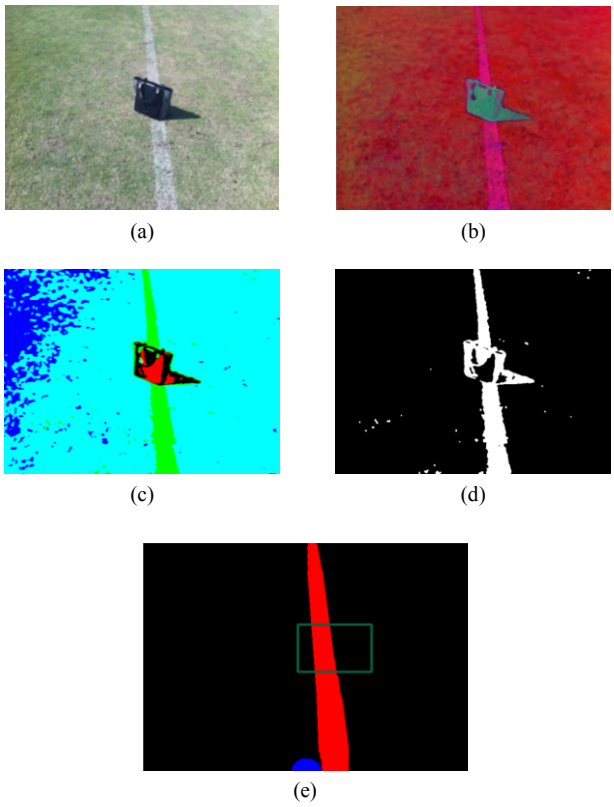


Fig. 4. Color space conversion from RGB to HSV and its effects on line detection: (a) The original frame in RGB, (b) Frame in HSV color space, (c) The effects of HSV thresholding, (d) Line detection output map is obtained from the thresholded frame, (e) Final resultant output map after applying Hough transform.

was followed in this work is inspired by the process outlined in [9]. The outcome of this data fusion process will yield a common occupancy grid. This grid is a combination of the occupancy grid that delineates the boundary lines and the occupancy grid that marks the obstacles. Having both lines and obstacles in one single map enables the UGV controller to perform path planning.

C. Path Planning and Tracking Module

The occupancy grid, discussed above, holds valuable information about the environment surrounding the UGV. In addition, the module accepts the following inputs:

- **GPS**: gives the current location of the UGV
- **Compass**: yields the UGV's heading (orientation)
- **Accelerometer**: used to detect vibrations caused by travelling over unleveled terrains.
- **Emergency stop (E-Stop)**

The module discussed in this subsection is responsible for processing all these inputs and, consequently, making the right decision in terms of UGV's speed and direction.

This module consists of two main processing blocks that operate hand-in-hand. These two blocks are the "*Path*

Planning" block and the "*Artificial Intelligence*" block. Fig. 5 depicts a block diagram that details the components of each block.

The *Path Planning* block finds an optimal route – in a clutter – to the destination. This block consists of three sub-blocks; namely, *Goal Finder*, *Weight Assignment*, and *Path Searcher*. The role of the *Goal Finder* is to find a goal in a 2D grid map given the GPS waypoints. The *Weight Assignment* takes a 2D grid and creates a new weighted 2D grid map. This map will have different weights assigned to each location on the grid. Assigning a higher weight to a location on the grid map prohibits the UGV from travelling across that location, and vice versa. Finally, *Path Searcher* uses the weighted 2D grid map to find the optimal path that the vehicle should take. This optimal path is represented by an array of (x, y) coordinates, which are passed to the *Artificial Intelligence* block for further processing.

To elaborate on the process explained above, consider the scenario shown in Fig. 6.a. In this scenario, a 2D grid map represents the environment surrounding the UGV. After reading the inputs, the destination will be found and marked on the grid. Thereafter, cost will be assigned to each cell in the grid. Then, A* algorithm is invoked to find the optimal path as shown in the figure, A special case is illustrated in Fig. 6.b, where the path is completely blocked by obstacles. In such a case, the UGV is forced to cross the boundary lines in order to reach the destination.

The *Artificial Intelligence* block employs fuzzy logic techniques in order to decide on the speed and direction of the UGV. In addition to the array of coordinates that are generated by the *Path Searcher*, the decisions made by this block are affected by the readings received from the compass, the emergency stop, and the accelerometer. This block is comprised of four sub-blocks. These are: *Fuzzification*, *Rule Base*, *Decision Making*, and *Defuzzification*. The role of *Fuzzification* is to convert the crisp input data to a fuzzy set. This fuzzy set is used by the *Rule Base* to generate rules, where each rule is assigned a weight. These weights indicate the ranking of the rules; where higher weights signifies greater importance. *Decision Making* is responsible for firing the best rule for a certain input data. Finally, *Defuzzification* converts the fuzzy output into crisp data, which is represented in a format that is understood by the UGV's motor controller.

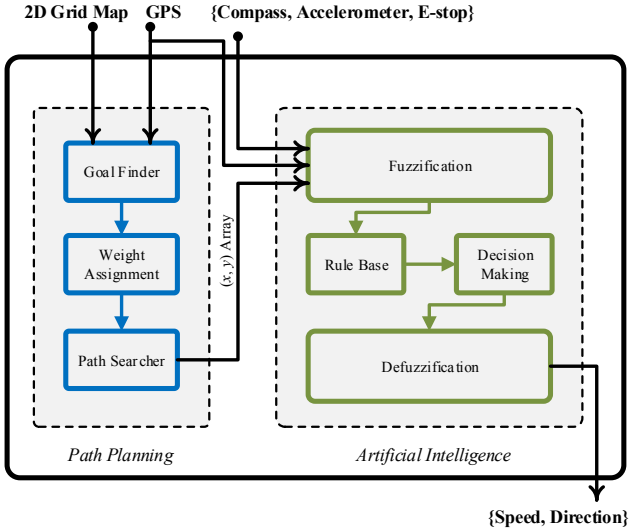


Fig. 5. Path Planning and Tracking Module. The Path Planning block finds the optimal path to the destination, while the *Artificial Intelligence* block computes the speed and direction of the UGV.

D. Real-Time Motor Driving Processing Module

This module is responsible for converting the speed and direction commands received by the path planning and tracking module into motor speed commands. These commands depend on the driving method that is used in the vehicle. In this work, differential drive method is adopted, where two wheels mounted on a common axis are controlled by separate motors. Controlling the speed of the left and right motors will control the vehicles speed and direction. The following kinematic equations are used in this module to calculate the desired speeds of the wheels:

$$x(t) = \frac{1}{2}\frac{v_r + v_l}{v_r - v_l}\sin\left(\frac{t}{l}(v_r - v_l)\right) \quad (1)$$

$$y(t) = \frac{1}{2}\frac{v_r + v_l}{v_r - v_l}\cos\left(\frac{t}{l}(v_r - v_l)\right) \quad (2)$$

$$\theta(t) = \frac{t}{l}(v_r - v_l) \quad (3)$$

here t is the goal time to get to positions (x, y, θ), v_r is the speed of the right motor, v_l is the speed of the left motor, and l is the distance between the centers of the two wheels**.** After calculating the desired speeds, this module generates a string which contains the speed of each motor and sends it to the motors' driver. The motor driver in turn provides the required voltage and current to both motors accordingly.

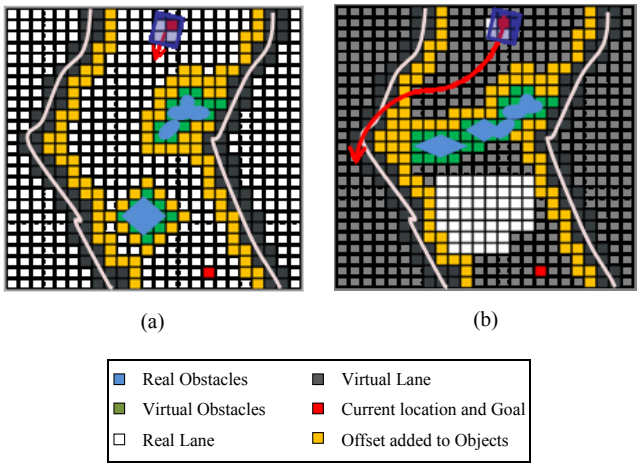


Fig. 6. Examples of real-world scenarios: (a) 2D grid map of a scenario where the UGV can maneuver within the boundary lines, (b) 2D grid map of an environment where the path is completely blocked by obstacles

IV. UGV System Integration

Autonomous UGVs are complex mobile robots housing a number of processing modules. Usually these modules are developed separately and different development environments might be utilized to develop each individual module. Take for example the processing modules presented in this work; where the line detection submodule was developed in OpenCV, while the obstacle detection and map making submodules were developed in LabVIEW. The choice of development environment may vary, since developers may have their own preferences. In addition, one development environment may offer toolboxes and APIs that are not available – or not as efficient as – the ones in another development environment. This dissimilarity in development environment creates a burden when it is time to integrate all submodules into a single unified system.

The approach that we followed in this paper is to develop a framework that is flexible enough to incorporate these individual submodules into a single unified platform. The following subsections discuss this framework.

A. Software Integration Framework

The output format of the developed processing modules depends on the nature of the development environment and the platform at which these modules were developed. For instance, an object file might be generated out of a processing module, where the former can be used later to be linked with other object files. Alternatively, executable files might be generated out of these modules. These executables can be run – on demand – by the main entity. Another approach is to transform these modules into libraries. These libraries can be either static or shared. In a static library scenario, the library is combined with the calling entity at compile time to produce a single executable file. However, in the case of a shared library, the library is called and utilized on demand, rather than combining the library with the calling entity. We discuss each of these approaches in the following subsections.

1) Modules as Object Code

Compilation translates the sources code into a language that is understood by the computer. This language, also known as machine language, is platform dependent. This dependency on the platform makes it difficult to integrate modules that were developed in different platforms (e.g. different target machines or operating system). To explain this further, let us use the same example as above, where line detection and obstacle detection submodules were developed using OpenCV and LabVIEW, respectively. In this case, there is no way – which we are aware of – to use the object file produced by OpenCV C++ code and combine it with the LabVIEW code. For this reason, this approach was excluded.

2) Modules as Executabes

Another approach is to produce an executable file out of each module. This executable file should target the Instruction Set Architecture (ISA) of the computer that will host all the modules. Then, the main entity will call these executables when needed.

While this approach is feasible, however, it is inefficient. The reason being that calling an executable in the platform environment has a high overhead that slows down the execution time – making this approach much less attractive.

3) Modules as Link Libraries

The third approach is to transform the modules into libraries. These libraries need to be in a format that is compatible with the target computing machine. One alternative is to produce these modules as static libraries that will be combined with the other modules at compile time. However, we do not favor this options for two reasons. First, combining static libraries into the main code bloats the size of the executable file, which makes loading the executable into memory much slower. Second, memory management of this executable might become a burden on the operating system of the target machine.

4) Modules as shared library:

Another alternative is to transform the modules into shared libraries. The generation of such libraries can be done by passing the proper options to the compiler. In Microsoft Windows platforms, the compiler will generate a Dynamic-Link Library file with (*.dll*) extension. In a Linux platform, the library will be in Shared Object format with (*.so*) extension. In either case, these shared objects are accessible through an Application Programming Interface (API), which exposes the functions implemented in the library to the outside world.

We choose this approach for two reasons:

- Shared libraries are loaded into memory on demand only (i.e. only when need by the main entity). This makes loading and unloading the modules an easier job and leads to more efficient usage of the system memory.
- The use of shared libraries maintains the modularity approach that we follow during the design of the autonomous UGV. To show the usefulness of shared libraries, let us assume that an improved version of the line detection submodule is now available. All what is required to incorporate this improved version into the framework is to recompile the new submodule. The main entity and all other submodules will remain intact and recompilation for these components is not required.

Fig. 7 illustrates the framework that was developed to incorporate all developed processing modules. The framework is flexible and extendable, where the inclusion and exclusion of processing modules (i.e. shared libraries) can be easily done and is transparent to the main entity. Another advantage of this framework is that it gives the freedom to developers to work in the platform and development environment of their preference. This is because the framework makes no previous assumptions on how these modules are developed.

B. *Outdoor Field Tests*

The UGV has undergone several field tests. In these tests, the UGV has correctly detected the boundary lines and dynamic obstacles. While travelling to a given destination waypoints, the UGV remained within the boundary lines and avoided multiple obstacles that were blocking the path. In summary, the UGV has successfully travelled its course from designated origin to destination.

V. Conclusions and Future Work

This paper presents a work-in-progress for developing the architecture and integration framework for an autonomous UGV. The UGV comprises sensing capabilities, processing modules, and a mechanical platform. The processing modules, which control the system, are capable of analyzing the environment around the vehicle and processing the gathered data to make an appropriate decision. The UGV was tested in an outdoor arena that is bounded by lines and obstacles. The vehicle has successfully travelled its course while avoiding obstacles.

For future work, the authors will continue to improve the platform. In particular, the UGV will be optimized to be able to navigate through more challenging environments, while maintaining real-time performance.

Acknowledgment

The authors would like to thank Nasser Hosseinzadeh, Head of the Department of Electrical & Computer Engineering and Abdullah Al Badi, Dean of the College of Engineering at the Sultan Qaboos University for their support. Ashraf Saleem thanks the following students for their role in developing the UGV mechanical platform and motor drive: Ahmed Al-shekaili, Ashraf Al-subaihi, Hadeel Al Sayed, Maadh Al Hinai, Juma Al Sabari, and Ismail Al Busaidi. Ahmed Al Maashri and Medhat Hussein would like to thank the students Abeer Al-Aghbari, Hiba Al-Farsi, Manea Al-Shukaili, and Sami Al-Abri for efforts in developing the path following and tracking modules and software framework. Lazhe Khriji thanks Khali Gamal, Ahmed Al Mahrouqi, Majda Al Falahi, and Rawana Hassanfor developing the image processing modules and LiDAR interface, and analysis subsystems.

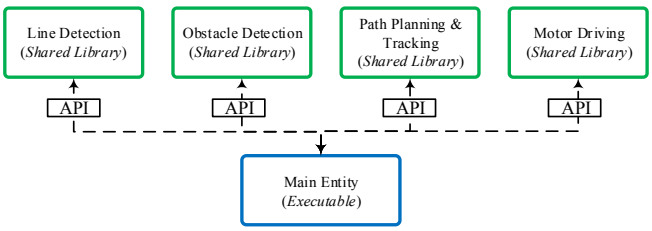


Fig. 7. The processing modules integration framework. The main entity is an executable that acts as an overarching main routine. The main entity orchestrates the invocation of processing modules, where it accesses these modules through a set of Application Programming Interface (API). This API exposes the functions/methods to the main entity. The processing modules themselves are shared libraries than can be either DLLs or SOs, depending on the target operating system. The framework is extendable, where processing modules can be added or removed, while the main entity remains intact.

References

[1] GlobalSecurity.org, Military Robots / Unmanned Ground Vehicles (UGV), [online]: http://www.globalsecurity.org/military/systems/ground/ugv.htm

[2] Faulhaber Global, Size is crucial, [online]: http://www.faulhaber.com/ch/markets/factory-automation-robotics/unmanned-ground-vehicle

[3] P. Papadakis. "Terrain traversability analysis methods for unmanned ground vehicles: A survey," in Engineering Applications of Articial Intelligence, vol. 26, Issue 4, April 2013, pp. pp.1373 – 1385, 2013.

[4] J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal "A Multi-Stage Optimization Formulation for MPC-Based Obstacle Avoidance in Autonomous Vehicles Using a LIDAR Sensor," in Proceedings of ASME 2014 Dynamic Systems and Control Conference, Texas, USA, October 22–24, 2014.

[5] G. Zhang, C. A. Duncan, J. Kanno, and R. R. Selmic "Unmanned Ground Vehicle Navigation in Coordinate-Free and Localization-Free Wireless Sensor and Actuator Networks," in Journal of Intelligent & Robotic Systems, vol. 74, Issue 3-4, 2014, pp. 869-891.

[6] P. Velaskar, A. Vargas-Clara, O. Jameel, and S. Redkar "Guided Navigation Control of an Unmanned Ground Vehicle using Global Positioning Systems and Inertial Navigation Systems," in International Journal of Electrical and Computer Engineering (IJECE), vol. 4, Issue 3, 2014, pp. 329-342, 2014.

[7] army-technology, "Oshkosh TerraMax Unmanned Ground Vehicle Technology," [Online]: http://www.army-technology.com/projects/oshkosh-terramax-unmanned-ground-vehicle/.

[8] Segfault team – Dalhousie University, "Robot Design Report," [online]: http://www.igvc.org/design/2013/Dalhousie%20University.pdf

[9] R. R. Murphy, "Introduction to AI robotics," The MIT Press, 2000.

[10] E. W. Dijkstra, "A note on two problems in connection with graphs," in Numerische Mathematik pp. 269–271, 1959.

[11] C. E. Leiserson and T. B. Schardl, "A Work-Efficient Parallel Breadth-First Search Algorithm," in ACM Symposium on Parallelism in Algorithms and Architectures, 2010.

[12] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics SSC4, vol. 4, Issue 2, 1968, pp. 100–107.

[13] M. Likhachev, D. Ferguson, and G. Gordon, "Anytime Dynamic A*: An Anytime, Replanning Algorithm.," ICAPS, 2005.

[14] O. Khatib and J. F. Le Maitre, "Dynamic control of manipulators operating in a complex environment," in Proceedings of 3rd CISM-IFToMM Symposium, September 12-15 1978.

[15] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobilerobots," in IEEE Transaction on Robotics and Automation, vol. 7, Issue 3, 1991, pp. 278–288.

[16] S. Parasuraman, B. Shirinzadeh, and V. Ganapathy, "Mobile Robots Navigation," ISBN 978-953-307-076-6, Published: March 1, 2010.

[17] H. Maaref and C. Barret, "Sensor-based navigation of a mobile robot in an indoor environment," vol. 38, Issue 1, 2002, pp. 1–18.

[18] D. Gu and H. Hu, "Neural predictive control for a car-like mobile robot," in Robotics and Autonomous Systems, vol. 39, Issue 2, 2002, pp. 73–86.

[19] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.