An End-to-End Robot System for Warehouse Applications Using Controller Synthesis

Benjamín Tapia, Juan Delmastro, and Sebastián Zudaire

Instituto Balseiro, Universidad Nacional de Cuyo, Río Negro, Argentina,
{benjamin.tapia, juan.delmastro, sebastian.zudaire}@ib.edu.ar

Abstract. Controller synthesis has been used in recent research to generate, from user specifications, correct-by-construction motion and task plans for mobile robots. These plans are implemented by hybrid control architectures allowing the robot to move and interact with complex environments. However, the different abstraction levels involved together with the many components that must be developed for a fully functional end-to-end system requires numerous design and methodology decisions, specially when the robot strongly interacts with its environment. In this paper, we present an end-to-end system design and implementation for a mobile robot targeted at warehouse applications, that uses a proof-of-concept approach in a simulated environment as a key step in the design process. We demonstrate its capabilities in product rearrangement both in simulated and real world scenarios.

Keywords: discrete events, hybrid control, warehouse applications

1 Introduction

Autonomous robots are increasingly being used in domestic and industrial environments to perform a variety of different tasks ranging from home cleaning [27] to car manufacturing in assembly lines [6]. For the past decades, the Robotics and Automated Software Engineering community have extensively studied how to specify and generate motion and task plans for these robots in order to interact with complex reactive environments [5, 13, 14, 21–23].

The field of controller synthesis has provided a number of tools [8, 10, 29] to automatically generate correct-by-construction plans for robot missions from user-supplied specifications, that model the robot capabilities and its environment, and express the system's goal. To implement these high-level and discrete synthesised controllers in real-world applications several hybrid control architectures have been proposed to interface with the low-level continuous control-loops and actuators of the robot [3, 17, 21, 31].

As a result, numerous end-to-end implementations of mobile robots [13, 21] have been developed allowing for a user to specify missions in some formal language, such as Linear Temporal Logic, and synthesise a plan. This plan is then translated into continuous movements and actions of the robot that guarantees to satisfy the mission specification if a set of assumptions hold. However, there are

many different methodological and design decisions that are made throughout these implementations that have led to discussions about how these systems can be designed [1, 7]. Missions that involve precise interaction with the environment as in [21] may require abstracting the robot's capabilities into control-modes [2], while others focused on movement can be abstracted from the robot's dynamics as in [4]. Thus, the domain of application influences the way the hybrid system may be designed.

In this work, we explore this topic for the particular scenario of mobile robots for warehouse applications (e.g., [30]). Hybrid control implementations for this scenario are, to the best of our knowledge, limited to simulated environments. Because we consider this level of abstraction fails to capture the complexity behind the robot-environment interactions to accommodate products in storage, it is our goal to develop and implement a solution into a real-world scenario, leveraging ideas from the design methodology presented in [19].

The main contributions of this work are: (1) an end-to-end hybrid robot system design and construction suitable for warehouses tasks validated in simulated and real-world mission scenarios, (2) a novel proof-of-concept approach in simulation during the design process of the hybrid control system to validate the correct behaviour of the proposed solution.

We structure the paper as follows. Section 2 presents the preliminaries of planning and hybrid controllers, and Section 3 shows the targeted warehouse applications with our proposed actuation solution, for which we explain the design approach in Section 4. We describe the implementation of the hybrid control system in a simulated environment in Section 5 and later the changes we made to construct and validate the real system in Section 6. Finally, Section 7 concludes.

2 Background

Here we present briefly the most important concepts of the planning formalism we will use. In Section 5.1 we will show how this formalism may be used to synthesise a discrete event controller for the domain of application of warehouse missions.

Labelled Transition Systems (LTS). The dynamics of the interaction of a robot with its environment will be modelled using LTS [15], which are automata where transitions are labelled with controllable and uncontrollable events that constitute the interactions of the modelled system with its environment. Complex models can be constructed by *parallel composition* ($||$) of LTS.

Fluent Linear Temporal Logic (FLTL). In order to describe environment assumptions and system goals we will use the formal language FLTL [11], a variant of linear-time temporal logic that uses fluents to describe states over sequences of actions. A fluent fl is defined by a set of events that make it true ($Set_\top$), a set of events that make it false ($Set_\bot$) and an initial value (v) true ($\top$) or false ($\bot$): $fl = \langle Set_\top, Set_\bot, v \rangle$. We may omit set notation for singletons

(a) (b)

Fig. 1: (a) Entry movement to accommodate the robot under the rack, (b) Exit movement to load the product on the robot.

and use an action label ℓ for the fluent defined as $fl = \langle \ell, Act \setminus \{\ell\}, \bot \rangle$. Thus, the fluent ℓ is only true just after the occurrence of the action ℓ.

FLTL is defined similarly to propositional LTL but where a fluent holds at a position i in a trace π based on the events occurring in π up to i. Temporal connectives are interpreted as usual: $\Diamond\varphi$, $\Box\varphi$, and $\bigcirc\varphi$ mean that φ eventually holds, always holds, and holds on the next position of the trace, respectively.

Discrete Event Controller Synthesis. Given a LTS E with a set of controllable actions L and a task specification G expressed in FLTL, the goal of controller synthesis is to find an LTS C such that $E \| C$: (1) is deadlock free, (2) C does not block any non-controlled actions, and (3) every trace of $E \| C$ satisfies G. When goals are restricted to Generalized Reactivity (1) (GR(1)) the control problem can be solved in polynomial time [25]. GR(1) formulas are of the form $\bigwedge_{i=1}^{n} \Box\Diamond\psi_i \Rightarrow \bigwedge_{i=1}^{m} \Box\Diamond\varphi_i$ where ψ_i and φ_i are Boolean combinations of fluents that refer to assumptions and goals, respectively. In this paper we use MTSA [8] for solving control problems.

Hybrid Controller. In robotics, the difference between the continuous vs. discrete description of the real world, and the interaction between *discrete event controllers* and robot actuators and sensors requires a non-trivial translation task that is implemented in a *hybrid control layer* [3, 9].

3 Warehouse Applications

Recent developments in mobile robots have allowed them to carry out complex manipulation tasks such as trash pickup [24], dishwasher unloading [27], amongst others. Our focus is to include manipulation capabilities adequate for warehouse applications, which involve multiple item-deliveries [30]. There are numerous choices of manipulators, for instance, in [12] they use a 6 degree-of-freedom (DOF) arm, while in [26] a 4-DOF arm is proposed. However, simpler actuator solutions can be used to pick up objects of certain characteristics, such as the two robot handling strategy in [20].

For the purpose of this paper a simple actuator capable of loading and unloading a predefined set of box-shaped products in a single level storage will suffice. Our proposal of actuating mechanism consists of the combination of robot mobility, a locking/unlocking mechanism and carefully designed racks. These

racks consist of a pair of sliding surfaces that hold a single product on its sides, allowing the product to slide forward and backwards as part of the grab/drop mechanism. The top of the robot is designed to carry a single product thanks the design of a locking mechanism. In order to load the box onto the robot, controlled movements are used to carefully accommodate the robot under the rack and the product resting there (see Fig. 1a), activating the locking mechanism and with a reverse motion the product is mounted on the robot (see Fig. 1b). The unload sequence consists in the robot pushing the product on to the rack, unlocking and backing up, leaving the box on the rack.

4 Design methodology

Our design process is based on the work in [19], where an approach to constructing hybrid control mobile robots is presented. This methodology starts with the user providing a high-level mission specification from which plans can be automatically synthesised for the desired domain of application of the robot. This fixes an adequate abstraction level for which synthesis algorithms can compute a plan in reasonable time (seconds to minutes). Setting an abstraction level means that actions and events that must be implemented at the hybrid control layer are fixed, and we must then design taking them into consideration. Following the approach in [19] we would begin at this stage working on the real robot. However, we found that working first in a simulation environment can have many advantages which we will briefly describe.

The use of a simulated environment allows for quick testing and modification. This is particularly relevant in our target warehouse application since we have a potentially complex interaction between the robot and its environment. In the simulator we can evaluate the physical aspects of the robot and their relation to the control algorithms that must be developed. This means that we can test different feedback controllers and actuation policies to determine if they are adequate for the manipulation tasks. Validating with a *proof-of-concept* approach the system in a simulator with a good level of detail lets us proceed with confidence in the more costly real-world implementation.

The resulting design process is summarised in Fig. 2. Here we can see that we follow the methodology in [19] but adding a simulation environment to produce and validate a simulated prototype as *proof-of-concept*. This allows us to proceed into improving and adjusting the developed simulated components into the equivalent real-world components (shown in dashed red lines).

5 Simulated Environment

For our simulating environment we chose the *CoppeliaSim* simulator, a powerful tool which is widely used in robotic research for rigid body simulations [18]. Thanks to its architecture users can run custom code using LUA scripts that can be programmed inside the simulator interface or by using a remote API that can connect with code written on Python or C++. In this section we present

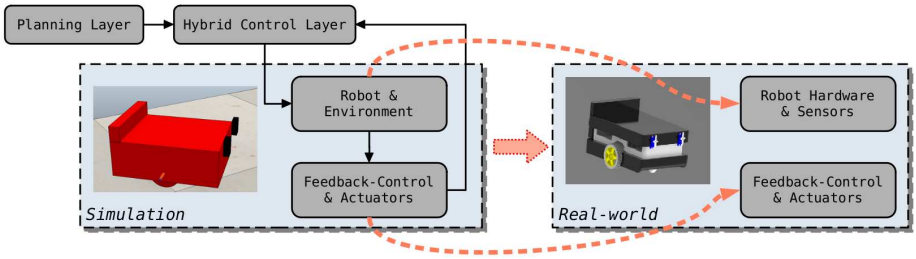
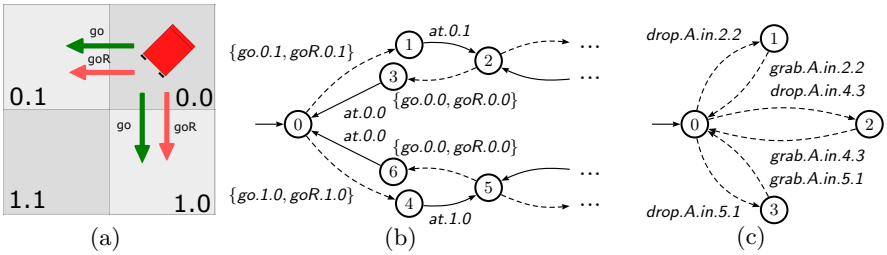


Fig. 2: Design process of our mobile robot for warehouse applications.



(a) (b) (c)

Fig. 3: (a) Environment cell discretization, and LTS models for (b) bidirectional movement, (c) grab/release capabilities. Dashed and continuous lines denote controllable and uncontrollable events, respectively.

the design and implementation of the robot using *CoppeliaSim* for the domain of application defined in Section 3.

5.1 Synthesis of Warehouse Missions

The work in [30] shows how warehouse missions may be synthesised from a GR(1) fragment of LTL specifications. However, we need to modify these specifications to adapt them from state-based to event-based specifications that we will express in a combination of FLTL and LTS, similarly as to how [19] adapts from [16].

The environment of this problem involves two main components: the movement capabilities and the grab/release mechanism to interact with the boxes. To produce a movement model we first discretize the working environment (see Fig. 3a) in adjacent cells as in [30]. Due to how our grab and release mechanism will work we include two different movement actions to go from one discrete location to the next: *go.i.j* to go in *front-first*, and *goR.i.j* to go in *backwards*. In Fig. 3b we show a portion of the LTS used to represent adjacent front and backwards movement capabilities between cells, which can be easily extended to the full discrete workspace as in [19].

To keep track of where the boxes are in the environment and to model the grab and release actions we use a LTS as the one shown in Fig. 3c, where we model that box A (initially on the robot) can be dropped in any cell from this set

$\{2.2, 4.3, 5.1\}$ with a *drop.A.in.i.j* action. Once dropped in a location it can only be grabbed at the same location with *grab.A.in.i.j*. Note that, as in the movement model, the initial state has to be set according to the initial conditions of the system (state 0 represents that the box is on the robot, state 1 that it is at location 2.2, etc).

The environment E is defined from the parallel composition of the LTS in Fig. 3b and 3c. However, this behaviour needs to be further restricted to correctly represent the desired behaviour of the overall system. Thus we include the following FLTL property to restrict that a box B must be at the respective $i.j$ location in order to be grabbed/released, where fluent $At.i.j$ is defined as $At.i.j = \langle\{\mathsf{at.i.j}\}, \{\mathsf{go.m.n}, \mathsf{goR.m.n} : m.n \in D\}, \bot\rangle$ for a cell discretization D:

$$\bigwedge_{i.j \in \{2.2, 4.3, 5.1\}} \Big[\Box(\mathsf{grab.B.in.i.j} \rightarrow At.i.j) \wedge \Box(\mathsf{drop.B.in.i.j} \rightarrow At.i.j)\Big] \quad (1)$$

We must also ensure that the robot will only access the shelves from a particular location, entering *front-first* and exiting *backwards*. Moreover, we will only allow *backwards* movement when exiting from a shelf. For this we include the following property for shelf-location $f.g$ that must only be accessed from $a.b$, assuming the shelves are in locations $\{2.2, 4.3, 5.1\}$ and where fluent $Last.i.j$ is defined as $Last.i.j = \langle\{\mathsf{at.i.j}\}, \{\mathsf{at.m.n} : m.n \in D \setminus \{i.j\}\}, \bot\rangle$:

$$\begin{aligned}\bigwedge_{i.j \in D} \Big[\Box\big(\mathsf{goR.i.j} \rightarrow (Last.2.2 \vee Last.4.3 \vee Last.5.1)\big) \wedge \Box\big(Last.f.g \rightarrow \neg\mathsf{go.i.j}\big)\Big] \\ \wedge \bigwedge_{i.j \in D \setminus \{a.b\}} \Big[\Box\big(Last.f.g \rightarrow \neg\mathsf{goR.i.j}\big)\Big] \wedge \Box(\mathsf{go.f.g} \rightarrow Last.a.b)\end{aligned} \quad (2)$$

Finally, for multiple boxes scenarios (e.g., two boxes A and B) we disallow entering a shelf in location $f.g$ that already has a box with another box on the robot, where the fluents are defined as $Robot.has.A = \langle\{\mathsf{grab.A.in.i.j} : i.j \in \{2.2, 4.3, 5.1\}\}, \{\mathsf{drop.A.in.i.j} : i.j \in \{2.2, 4.3, 5.1\}\}, \bot\rangle$ and $Box.A.in.i.j = \langle\{\mathsf{drop.A.in.i.j}\}, \{\mathsf{grab.A.in.i.j}\}, \bot\rangle$:

$$\begin{aligned}\Box\Big(\mathsf{go.f.g} \rightarrow \big((\neg Robot.has.A \wedge \neg Robot.has.B) \vee \\ (Robot.has.A \wedge \neg Box.B.in.f.g) \vee (Robot.has.B \wedge \neg Box.A.in.f.g)\big)\Big)\end{aligned} \quad (3)$$

The environment E together with this set of FLTL specifications determine the admissible behaviour of a robot in a warehouse scenario. Depending on the mission, the user may include additional properties. For instance, in order to organise the warehouse one can specify the final locations of two boxes A and B with the following property: $\Box\Diamond(Box.A.in.4.3) \wedge \Box\Diamond(Box.B.in.2.2)$. As we will show in Section 5.4, a controller satisfying this specification will make use of the available shelves to accommodate the boxes in the correct way.

5.2 Hybrid Control Layer Design

Once we can produce synthesised plans for our desired domain of application, we proceed to building the necessary hybrid control components in order to

translate the discrete actions in continuous signals for the robot actuators and feedback-controllers. The hybrid architecture of our system is very similar to the one shown in [19], so we will focus on describing the differences in the hybrid modules we implemented. We build two hybrid modules: *Motion control* and *Grab control*.

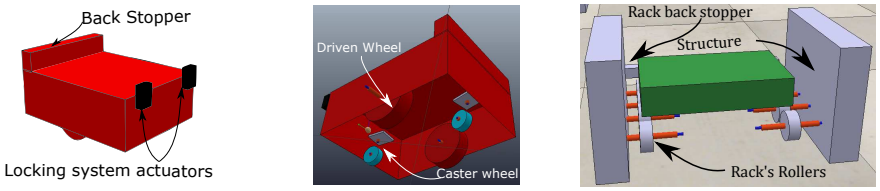
Motion control: Here we elaborated on the motion algorithms presented in [19] to include **simultaneous** rotating and forwards/backwards movement. This module takes as an input *go.i.j* or *goR.i.j* as a target position, and applies a control algorithm to generate continuous movement that leads the robot to this target location. For this we will use a combination of two controllers: one for the robot's orientation, by commanding the wheels in opposing directions and one for advancing movements front- or back-facing, depending on whether the robot is required to move forwards (*go.i.j*) or backwards (*goR.i.j*). Once a new target position is defined two references are calculated: distance from the target and the deviation angle from the correct orientation required to face the target. The orientation controller will align the robot with the objective position. Once the angle is within a given threshold, the forward/backward movement controller moves the robot in a unidirectional motion. In order to achieve smooth and robust trajectories, while moving forwards or backwards the orientation controller is activated to ensure that the robot will keep facing the target in the presence of disturbances. As shown in [19], an adequate low-cost mobile robot configuration to achieve this movement is a two-wheel robot.

Grab control: As described in Section 3, once the robot is positioned under a box A it has to lock or unlock the product from the robot. This module interprets the grab (*grab.A.in.i.j*) and drop (*drop.A.in.i.j*) commands and sets the positions of the lock system down to unload a product onto the rack or up to load the product onto the robot.

5.3 Implementation in Simulated Environment

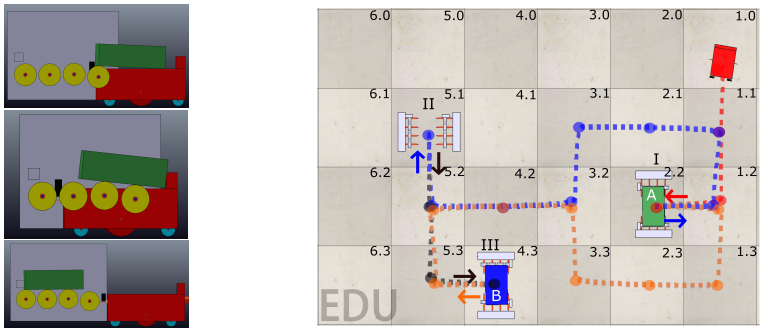
For realistic results we have to provide the simulator's software with important physical parameters such as weight, centre of mass and inertial moments. These were obtained from a preliminary 3D CAD model for the mobile robot using *Solid Edge*. This model was created for the following parameters: (a) two-wheel drive with two caster wheels for stability (See Fig.4b), (b) enough room to layout all the standard electronic components that will be required, (c) a flat top surface of an adequate size to carry a box-shaped product of $15 \times 25 \times 5$cm.

We then recreated the design of the robot in *CoppeliaSim* (see Fig. 4a) by incorporating the physical parameters measured in the CAD model. For the locking mechanism and wheels, revolute joints were used and configured for motor and position control, respectively. For loading and unloading of products we need to define the sliding surfaces on the racks and the locking mechanism of the robot. As sliding surfaces we used same sized rollers at different heights to both produce minimal movement effort from the robot when unloading by offsetting the first roller to a lower height. To ensure that the product will stay



(a) (b) (c)

Fig. 4: (a) Modelled robot (top view), (b) Wheel distribution, (c) Rack model.



(a) (b)

Fig. 5: (a) Product loading sequence, (b) Validation in simulated environment.

in place once it's on the rack, a small slope is created with the rollers. The resulting rack model can be seen in Fig.4c.

The locking/unlocking mechanism consists of a fixed back stopper on the robot and movable front servos (see Fig.4a). When the front servos are upwards, the product will be restricted in its longitudinal movement and surface friction between the product and the top of the robot will prevent it from moving sideways. When the front servos are in the down position all movements are only restricted by surface friction.

Unloading of products can be achieved with the following steps: (1) the robot approaches the rack with the product and the servos in the up position, and begins pushing the box onto the rack with ease thanks to the first lower roller, as shown in the top of Fig. 5a, (2) as the robot continues to push the box onto the rack, the back stopper will prevent the box from falling off the back (see middle of Fig. 5a), (3) once the box is completely inside the rack, the robot can lower the servos and exit backwards, while the box stays in place thanks to a small slope towards the end and given that the bottom of the product is now higher than the top of the robot (see bottom of Fig. 5a). The steps to load a product from the rack onto the robot are similar but in a reverse sequence with the robot going in with the servos down and leaving with the servos lifted.

One of the advantages of working with a simulator is that we have direct access to all the physical variables of the simulation so we can simplify the

implementation process at this stage by not including sensors altogether. For a *proof-of-concept* approach this is reasonable if we consider at all times how we will measure each variable in the real world. For instance, the *Motion control* algorithm requires continuous readings of the position and orientation of the robot that can be obtained from an aerial view of the workspace.

We then proceeded to implement the angle and movement controllers required by the *Motion control* module, as well as the lock/unlock implementations of the *Grab control* module, in the simulator using LUA scripts. The rest of the hybrid control architecture was implemented using the remote API interface communicating with a custom Python code. This separation over two programming languages was made to maximise re-usability of the code when switching to the real world, as we will describe in Section 6.2. The lock/unlock implementations can be achieved easily by commanding the respective servos to two different target angles. To implement the more complex angle and movement controllers we worked with similar feedback and open-loop controllers as in [19].

5.4 Validation in the Simulated Environment

For validation we will use the workspace partially shown in Fig. 5b, a 7×7 grid-partitioned environment. To evaluate the correct behaviour of the overall system we considered a single-robot variant from the mission in [30]: accommodate two products in their respective racks from the three available (I, II and III) as shown in Fig. 5b. Controller-synthesis and simulations were run on a Intel i7-8550U 1.80 GHz CPU with 8 GB RAM. The mission specification, Python codes, *CoppeliaSim* models (loaded with custom LUA scripts) and mission video of the simulated run can be found in [28].

The goal of the mission can be expressed with the FLTL formulas as in 5.1, where boxes A and B must be accommodated in the racks in II and I at 5.1 and 2.2, respectively. The position of the racks and how they can be accessed (see arrows in Fig. 5b) can modelled as shown in Section 5.1. For this specification, a discrete-event controller was synthesised in less than 3 s. Once synthesised, the simulation was started and the mobile robot proceeded to implement the task plan, as shown in Fig. 5b.

The coloured dashed lines indicate the robot's path as it completed the tasks. The robot first goes to rack I (red path) and grabs the green box A through the front-facing location 1.2. Then, it takes this box to position 2.2 and proceeds to unload it onto rack II (blue path). Now the robot is free to grab another product, so it goes to rack III (black path), loads box B and takes it to rack I (orange path), achieving the goal of the mission. The coloured dots on the figure represent the moment where the events *go.i.j* or *goR.i.j* were triggered. The full mission duration was around 90 s. Note that all decisions (path taken, which box to take first, etc) were determined by the synthesis algorithm which is correct-by-construction regarding the specification.

From this simulation we can conclude that the movement precision of the robot is adequate to perform grab and drop actions without any collision with the rack, as well as that the rack's design allows for satisfactory load and unload

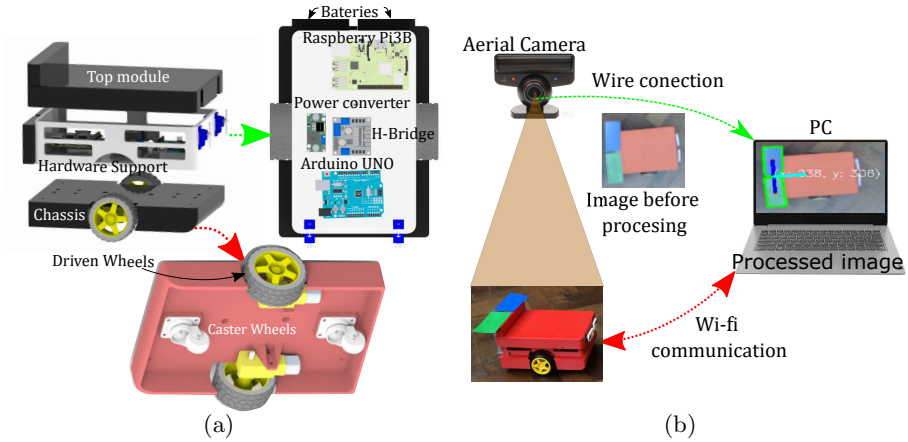


Fig. 6: (a) Robot modular design, (b) Camera sensing setup.

of boxes. Moreover, this experiment shows a correct execution of the complete hybrid controller in a representative mission scenario for warehouse tasks. We can thus proceed to build the system in the real world with more confidence on the proposed solution.

6 Real World

In this section we will discuss all the challenges involved in a real world implementation, as well as details on the constructed platform.

6.1 Platform Design and Hardware

The mobile platform consists of three modules, as shown in Fig. 6a: the first is the chassis of the robot, where the wheels (casters and driven) are mounted, the second module is in the middle and holds all the electronic components, and the third top module supports the box between the back stopper and the servos, as part of the locking/unlocking mechanism. All the structural components of this design were 3D-printed with a PLA polymer.

The electronics mounted on the robot (see top right of Fig. 6a) consist of: an Arduino Uno where the feedback and open-loop controllers are programmed, a Raspberry Pi 3B+ where the communication between a PC and Arduino is made, a L298N H-bridge to command the driven wheels, a power converter for the Raspberry Pi and an alternative converter for the rest of the electronic components, two MG90S servos for the locking mechanism and a set of batteries for the power supply of all the components

6.2 System Construction and Implementation

The implementation in the real world requires adjusting several of the components that were developed in the simulator. The hybrid controller architecture,

together with the control and planning algorithms may be the same, but they must be now implemented between three different interfaces: an Arduino, the Raspberry Pi, and an auxiliary PC.

The Arduino Uno is used to implement the feedback and open-loop controllers that were implemented as LUA scripts in the *CoppeliaSim* simulation. The auxiliary PC serves a similar purpose as that shown in [19], where in order to locate the robot in the environment, aerial images are captured and processed in this computer and later sent via Wi-Fi (see Fig. 6b) to the Raspberry Pi, which acts as an intermediary between the PC and the Arduino. Since information used by the continuous controllers at the Arduino level is extracted from images, the measurements sample time is limited by the FPS of the aerial camera, which we set to 20 Hz. Controller synthesis and the implementation of the higher hybrid levels could be done both in the Raspberry Pi or the PC, the latter being chosen for our work with Python 3.7.7 as the programming language, where we reused the Python code which interfaced with *CoppeliaSim* as described in Section 5.3.

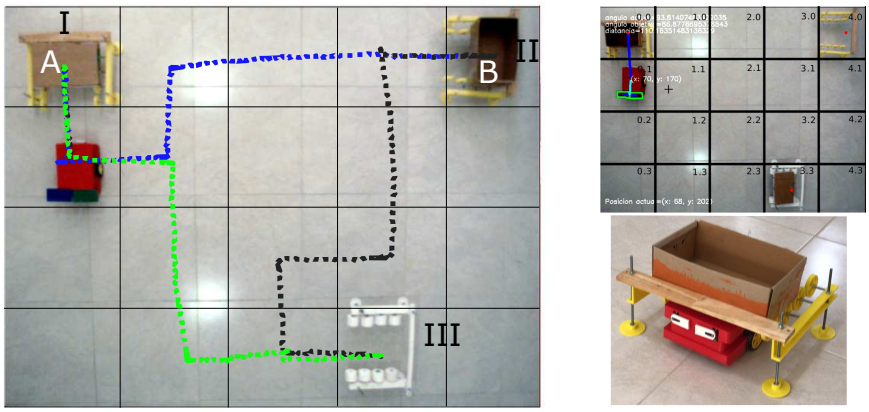
Communication between the Arduino and the Raspberry Pi is done through serial ports, where data such as the references of the feedback-controllers and the measured parameters of the robot (angle and distance to the objective position) are exchanged. This information is obtained from the aerial images using several functions from the OpenCV Python library. The image processing consists in identifying two different colour markers placed on the robot. This way we are able to trace a vector between them and know the angle and position of the robot. We do this by first implementing a mask over the original image to isolate the two markers. Once these are identified, we can obtain their contours and centre (see PC in Fig.6b). Finally, knowing the relative position between the two markers and the robot's centre we are able to determine its position and orientation in the environment.

The control algorithms that worked in the simulator showed to be not as robust as expected for real world implementation, giving oscillatory trajectories not suitable for the drop and grab actions, where a precise movement to enter the rack is required. For this reason we iterated over the implemented movement algorithm, including a more sophisticated proportional and derivative controller for the orientation.

6.3 Validation in the Real Environment

In this section we will validate the correct behaviour of the overall system by running a similar mission to the one presented in Section 5.4. For this, we will use the workspace shown in the top image in Fig. 7b, which consisted of a 2.5×2m environment partitioned in a 5×4 grid. Our goal is to accommodate boxes A and B in racks II and I (see bottom image in Fig. 7b) at 4.0 and 0.0, respectively. The mission was specified as in Section 5.4, and a discrete-event controller was synthesised in less than 1 s. The video as well as the Python codes and task specification of this mission are provided in [28].

We show in Fig. 7a the robot's trajectory as it executed its mission. As in the simulated validation, the coloured dashed lines indicate the robot's path as



(a) (b)

Fig. 7: (a) Aerial image with the robot's path, (b) Top: Processed image with grid partitioned environment, Bottom: Robot loading box in rack I.

it completed the task. The robot first goes to rack I (red path) and grabs the box A. Then it proceeds to take this box to position 0.0 and proceeds to unload it onto rack II (blue path) at 4.0 position. Now the robot goes to grab the box B in rack III (black path) at 3.3 position and delivers it to rack I (green path) at 0.0, achieving the specified task. The full mission duration was around 90 s.

From real world implementation we can conclude that the movement and the locking mechanism of the robot are adequate and work correctly for the precise movements required for loading and unloading of products on the racks. Moreover, we validated the correct behaviour of the end-to-end system in a representative task of warehouse scenarios.

7 Conclusions and Future Work

In this work we presented an end-to-end robot system that successfully accomplishes warehouse applications tasks, synthesised from from high-level specifications. For this, we proposed a design methodology, where we first worked in a simulated environment to produce and later validate a simulated prototype as a proof-of-concept. This allowed us to proceed with a detailed design for the real-world implementation. Finally, we validated the correct behaviour of our end-to-end system in a representative task of warehouse scenarios taken from the literature, accommodating two products between three racks. In future work, it would be interesting to analyse how the proposed platform scales with multi-robot interactions in larger environments, as well as modifying the item loading capabilities to support multiple levels of storage.

References

1. Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: Discrete abstractions of hybrid systems. Proceedings of the IEEE **88**(7), 971–984 (July 2000)
2. Balkan, A., Vardi, M., Tabuada, P.: Mode-target games: Reactive synthesis for control applications. IEEE Transactions on Automatic Control **63**(1), 196–202 (2018)
3. Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., Pappas, G.J.: Symbolic planning and control of robot motion [grand challenges of robotics]. IEEE Robotics Automation Magazine **14**(1), 61–70 (2007)
4. Belta, C., Habets, L.C.G.J.M.: Constructing decidable hybrid systems with velocity bounds. In: 2004 43rd IEEE Conference on Decision and Control (CDC). vol. 1, pp. 467–472 Vol.1 (2004)
5. Braberman, V., D'Ippolito, N., Kramer, J., Sykes, D., Uchitel, S.: Morph: A reference architecture for configuration and behaviour self-adaptation. In: Proceedings of the 1st International Workshop on Control Theory for Software Engineering. pp. 9–16. CTSE 2015, ACM, New York, NY, USA (2015)
6. Chang, K., Xing, Y., Yin, J., Zheng, D.: Design of intelligent management and control system of flexible transmission assembly line. In: 2018 Eighth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC). pp. 756–760 (2018)
7. DeCastro, J.A., Raman, V., Kress-Gazit, H.: Dynamics-driven adaptive abstraction for reactive high-level mission and motion planning. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). pp. 369–376 (2015)
8. D'Ippolito, N.R., Braberman, V., Piterman, N., Uchitel, S.: Synthesis of live behaviour models. In: Proceedings of the 18th ACM SIGSOFT Symposium on the Foundations of Software Engineering. pp. 77–86. USA (2010)
9. Fainekos, G.E., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for mobile robots. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. pp. 2020–2025 (2005)
10. Finucane, C., Gangyuan Jing, Kress-Gazit, H.: Ltlmop: Experimenting with language, temporal logic and robot control. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1988–1993 (Oct 2010)
11. Giannakopoulou, D., Magee, J.: Fluent model checking for event-based systems. In: Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 257–266 (2003)
12. Gong, W., Xu, H., Li, Q., Gu, H., Han, C., Song, S., Meng, M.Q.H.: Mobile robot manipulation system design in given environments. In: 2017 IEEE International Conference on Information and Automation (ICIA). pp. 609–613 (2017)
13. Jing, G., Tosun, T., Yim, M., Kress-Gazit, H.: An end-to-end system for accomplishing tasks with modular robots: Perspectives for the ai community. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. pp. 4879–4883 (2017)
14. Kaelbling, L.P., Lozano-Pérez, T.: Hierarchical task and motion planning in the now. In: 2011 IEEE International Conference on Robotics and Automation. pp. 1470–1477 (May 2011)
15. Keller, R.: Formal verification of parallel programs. Communications of the ACM **19**, 371–384 (07 1976)
16. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Trans. on Robotics **25**(6), 1370–1381 (2009)

17. Kress-Gazit, H., Fainekos, G., Pappas, G.: Translating structured english to robot controllers. Advanced Robotics **22**, 1343–1359 (10 2008)
18. Kumar, P.B., Verma, N.K., Parhi, D.R., Priyadarshi, D.: Design and control of a 7 dof redundant manipulator arm. Australian Journal of Mechanical Engineering pp. 1–12 (2019)
19. Liendro, T., Zudaire, S.: Hybrid control from scratch: a design methodology for assured robotic missions. In: Simposio Argentino de Inteligencia Artificial - JAIIO 49. pp. 1–14 (2020)
20. Liu, Z., Kamogawa, H., Ota, J.: Manipulation of an irregularly shaped object by two mobile robots. In: 2010 IEEE/SICE International Symposium on System Integration. pp. 218–223 (2010)
21. Maniatopoulos, S., Schillinger, P., Pong, V., Conner, D.C., Kress-Gazit, H.: Reactive high-level behavior synthesis for an atlas humanoid robot. In: 2016 IEEE International Conf. on Robotics and Automation (ICRA). pp. 4192–4199 (2016)
22. Menghi, C., Tsigkanos, C., Pelliccione, P., Ghezzi, C., Berger, T.: Specification patterns for robotic missions. IEEE Trans. on Soft. Engineering pp. 1–1 (2019)
23. Nedunuri, S., Prabhu, S., Moll, M., Chaudhuri, S., Kavraki, L.E.: Smt-based synthesis of integrated task and motion plans from plan outlines. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). pp. 655–662 (May 2014)
24. Pattanashetty, K., Balaji, K.P., Pandian, S.R.: Educational outdoor mobile robot for trash pickup. In: 2016 IEEE Global Humanitarian Technology Conference (GHTC). pp. 342–351 (2016)
25. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. Lecture notes in computer science **3855**, 364–380 (2006)
26. Rodriguez Lera, F., Casado Garcia, F., Esteban, G., Matellan, V.: Mobile robot performance in robotics challenges: Analyzing a simulated indoor scenario and its translation to real-world. In: 2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation. pp. 149–154 (2014)
27. Saxena, A., Driemeyer, J., Ng, A.Y.: Robotic grasping of novel objects using vision. The International Journal of Robotics Research **27**(2), 157–173 (2008)
28. Tapia, B.: Task specifications, simulation and videos, https://bitbucket.org/BenjaTapia/an-end-to-end-robot-system-for-warehouse-applications-using, , accessed 2021-08-22
29. Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., Murray, R.M.: Tulip: A software toolbox for receding horizon temporal logic planning. In: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control. pp. 313–314. HSCC '11, ACM, New York, NY, USA (2011)
30. Yang, Y., Zhao, J., Yin, X., Li, S.: Formal synthesis of warehouse robotic systems with temporal logic specifications. In: 2019 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM). pp. 469–474 (2019)
31. Zudaire, S., Garrett, M., Uchitel, S.: Iterator-based temporal logic task planning. In: 2020 International Conference on Robotics and Automation (ICRA). pp. 11472–11478 (2020)