An Abstraction Layer for Controlling Heterogeneous Mobile Cyber-Physical Systems

TREVOR HANZ
Department of Computer Science
Texas State University
th1382@txstate.edu

MINA GUIRGUIS
Department of Computer Science
Texas State University
msg@txstate.edu

***Abstract*— Mobile Cyber-Physical Systems (CPSs) widely vary in the underlying hardware technologies and capabilities of their mobile devices (e.g., robots). This makes the problem of developing portable high-level Mobile CPS applications difficult, since applications are typically written for specific devices. To this end, this paper present a framework that relies on abstracting the representation of the mobile devices in a manner that allows the system to fully utilize the capabilities of the hardware while providing a simplified interface to the end-user that can work across different devices. This framework incorporates a physics engine to handle different physical models of different mobile devices to ensure better control. We assess the behavior of our proposed framework through real experiments conducted in our Mobile CPS lab with various iRobot Creates. We show that we can achieve better control accuracy through exploiting the physical characteristics of the mobile devices, rather than relying solely on the driver interface of the devices.**

I. INTRODUCTION

Cyber-Physical Systems (CPSs) are systems that closely integrate computation with their physical environments through various communication mediums. Mobile CPSs is a subset of CPSs in which a subset of their components are mobile. Due to recent advances in wireless networking and embedded systems, mobile CPSs are emerging as powerful systems in various areas such as autonomous vehicles and swarm robotics. For example, many mobile CPS applications have been developed for exploration [1], [2], border control [3]–[5], and search and rescue operations in land [6], [7], water [8], and air [9], [10].

Many mobile CPS applications rely on devices being aware, to some extent, of their position in the physical world. While in some cases, the position can be obtained through external localization services (e.g., GPS or infrared), such systems may not be present and may not work in specific environments. Thus, devices must rely on sensors that are typically attached to their wheels and drive train (or accelerometers and gyroscopic sensors in the case of aerial devices). Moreover, these devices are typically equipped with various infrared, ultrasonic and visual sensors that are used to detect distances to physical objects, helping the devices to navigate and generate their relative positions in their environments. As one would expect various mobile devices are equipped with different types of sensors, capabilities and mobility models.

Currently the development of mobile CPS applications face a daunting portability challenge: applications designed (and implemented) using a particular hardware platform will not be able to run on another. This is due to the high dependency on the configuration of the hardware that the system is implemented on. For example, developing a robotics applications that can have a robot explore an area will depend on the type, dimensions, capabilities and the physical model of the robot. If one were to switch to a new robot the application must be redesigned and re-implemented. Although the new hardware must be taken into account, we argue that the high-level application may not need to change much to accommodate the new hardware. Since many tasks can be broken down into simple movement, rotation and sensing commands, the high level application can remain largely intact. Underneath, however, we must provide a mean to accommodate a heterogeneous set of mobile devices.

Devising control mechanisms for Mobile CPSs that are portable from a particular platform to another is a complex problem. It requires providing proper levels of abstractions that allow high level design and implementation to be reused, while supporting mapping common functionalities to different hardware platforms.

In this paper, we present a framework for managing mobile CPSs with different hardware configurations. The method relies on abstracting the representation of the mobile CPS – through physics models – in a way that can allow the system to utilize the capabilities of the hardware while providing a simplified interface to the end-user. Our proposed framework is particularly useful in Mobile CPS applications that utilize inexpensive mobile devices with low movement accuracy. Incorporating the physics models provides the proper forces and torques that are not typically captured with the traditional device drivers, leading to better accuracy. We have built a prototype of this framework as a proof-of-concept. Support for components that are not described in this paper may be easily built on top of this framework.

Paper organization: Section II describes our related work. In Sections III, IV and V we discuss the three components of our system, Core Framework, Physics Layer and Motion Control Layer, respectfully. The performance results are presented in Section VI followed by our conclusion and future work in Section VII.

II. Related Work

This work relates to different research areas for managing different mobile devices and enabling them to find their position in various environments. A lot of research has been done on finding the position of a mobile device and its relation to other objects. In [11] the authors describe seven types of positioning systems which includes odemetry as well as other methods that incorporate the use of more sophisticated hardware such as GPSs, magnetic compasses, and cameras. Many other experiments [12]–[14] have been conducted that implement and extend those methods using other hardware such as lidar and ultrasonic transceivers.

In order to manage this wide variety of hardware, a Robotic Operating System (ROS) is needed. In [15] a study is presented that gives an overview of the available ROSs including Microsoft's Robotics Developer Studio (RDS), Player/Stage, and the open source alternative that is named after what it is, ROS. This study started out surveying 16 different ROSs based on ease of use, capability, adaptability, ease of maintenance, and software maturity. The three mentioned above scored the highest in their study with ROS being selected as the most desirable.

ROS is a very robust system with many modules and drivers already developed including a driver for the Roomba/Create. This driver is a monolithic module that contains all functions necessary to access all capabilities of the Roomba. This includes controlling movement, reading sensor input and calculating odometry. While this driver has been proven to be quite useful, odometry calculations tend to be quite simplistic and often omit necessary information that is either harder to calculate or may not be accessible by the driver. For instance, the Roomba driver ignores acceleration as well as the friction between the wheels and the current surface the Roomba is on.

III. The Core Framework

The proposed framework aims to provide a subset of the more common capabilities, since it would be very difficult to support all possible capabilities of mobile devices. We have chosen to support a mobile device ability to understand its location and move to a new location. To support this we have implemented a system for odometry and an interface to the hardware to enable movements. It is provided by two components that are called the Physics Layer and Motion Control Layer. More details about these components can be found in the Sections IV and V, respectively.

While it is desirable to create a system with a simplified high-level interface, we did not want to remove any possible functionality from the mobile devices. To accomplish this, we decided to use a layered architecture as depicted in Figure 1. Each layer provides an additional level of abstraction from the layer below. There is no forced hierarchy so if a higher layer does not provide all the necessary functionality, a component may probe a lower layer to gather more detailed information. If possible, this should be avoided as lower levels tend to contain more system specific information which will generally make the accessing components less portable.

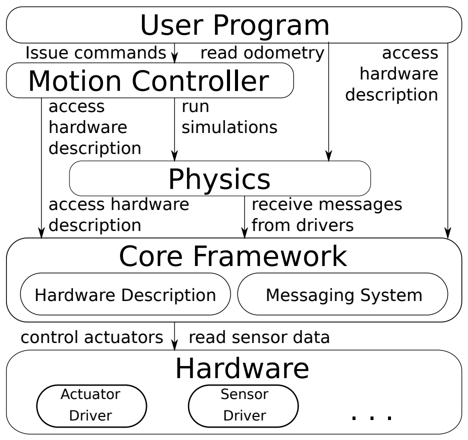

Fig. 1. Layered system architecture

This is also how the control system achieves easy extensibility. New components can be added to the system without affecting the existing components. A new component may access any of the other already existing components and in turn be accessed by other components to become a part of the system.

The lowest layer to which all components are, at least indirectly, connected is the core framework. The core framework contains the system specific information. It can be thought of like an operating system for the mobile device. Its job is to manage all known data about the hardware of the device as well as facilitate access to the drivers for the actuators and sensors. Additionally, the core framework provides a system for message passing between components.

The core framework manages data in a structured object-oriented manner. All data about the device is stored in objects that symbolically represent its physical components. The components are divided into three categories: structural components, sensors or actuators. Structural components define the physical shape of the device (e.g., wheels, chassis, etc...). They take parameters such as size, shape and weight. Sensors and actuators can be thought of as the input and output devices. Both of them provide access to the specific drivers necessary to utilize the hardware of the device. Sensors are periodically polled by the core framework at which point the sensor may broadcast a message about its current state to any listening process. Actuators can take a single floating point value normalized between -1.0 and 1.0. This value represents the desired state of the actuator. In the case of an electric motor, -1.0 to 1.0 would relate to full reverse to full forward, respectfully. It is done this way so that users of the actuators do not need to know the potential of the actuator to use its greatest potential. Whenever the actuator's state has changed, it should broadcast a message informing all listening processes about the change.

The components are then structured logically. The top level component is a structural component that represents the chassis and is referred to as the Agent. Some components, such as the Agent, may contain any number of other components allowing some components to be nested to any level. In addition to the properties mentioned earlier, all components contain a three dimensional position relative to their parent component with the Agent having an implicit position at the origin. It is structured this way to help other systems to better understand the relationship between components and the structure of the mobile device.

IV. Physics Layer

So far, a low level abstraction for the hardware has been established with the core framework. The next layer of our control system is the Physics layer. This layer provides a system for handling position and odometry measurements.

Odometry is often simplified for the specific hardware it is implemented on. For instance, the odometry calculations for a Roomba is depicted in Equations 1, 2 and 3 where θ is the Roomba's rotation, X and Y is the position of the Roomba in a Cartesian coordinate system representing the floor, k is a certain point in time and $\Delta(W_{Right})$ and $\Delta(W_{Left})$ is the change in the traveled distance of the Roomba's right and left wheels, respectfully.

$$\theta_k = \theta_{k-1} + \frac{\Delta(W_{Right}) - \Delta(W_{Left})}{(AxleLength)} \tag{1}$$

$$X_k = X_{k-1} + \frac{\Delta(W_{Right}) + \Delta(W_{Left})}{2} \times \cos\theta_k \tag{2}$$

$$Y_k = Y_{k-1} + \frac{\Delta(W_{Right}) + \Delta(W_{Left})}{2} \times \sin\theta_k \tag{3}$$

Since these equations would not work on other hardware except the Roomba, we used a more general system for calculating odometry. We use the information provided by the core framework to create a physics model. From there we sum up all the forces F and torques τ on each physical component and apply the constraints enacted on the physics model by the joints and the collision with the ground or other objects resulting in new force and torque vectors.

$$F_{total} = \sum F \tag{4}$$

$$\tau_{total} = \sum \tau \tag{5}$$

The new force and torque vectors can then be used in the calculation of the new linear and angular velocity.

$$v_k = v_{k-1} + \frac{F_{total}}{m} \times \Delta t \tag{6}$$

$$\omega_k = \omega_{k-1} + \frac{\tau_{total}}{m} \times \Delta t \tag{7}$$

Equations 6 and 7 depict the linear and angular velocity, respectfully, where v represents linear velocity, ω represents angular velocity, m represents mass and Δt represents change in time.

V. Motion Control Layer

So far as described, the control system is useful for providing information but it has no facilities to control the movements of the mobile device. For this reason, we developed the motion control layer. This system provides an additional level of abstraction on top of the physics layer and the core framework. It creates a high level interface to the actuators of the device so that users do not have to understand anything about its hardware to be able to control its movements.

This layer takes two three dimensional vectors representing linear and angular velocities as input and makes a decision on how best to move the available actuators in order to achieve the desired linear and angular velocities. The result of its calculations is an animation file describing the actuators to be moved and in what direction.

To create the animation file the system performs a linear regression using a simple batch gradient decent algorithm on each actuator. The batch gradient decent results in a model depicted in Equation 8 where X represents a six dimensional feature vector containing the desired linear and angular velocities and θ represents the correlation between the state of an actuator and the corresponding feature. This model must be repeated for each actuator after calculating a different θ vector for each actuator resulting in a model depicted by Equation 9 where n is the number of actuators.

$$\sum_{i=0}^{6-1} \theta_i \times X_i \tag{8}$$

$$\begin{array}{c} 0 \\ \vdots \\ n-1 \end{array} \begin{pmatrix} \sum_{i=0}^{6-1} \theta_{i,0} \times X_i \\ \vdots \\ \sum_{i=0}^{6-1} \theta_{i,n-1} \times X_i \end{pmatrix} \tag{9}$$

To calculate the θ vector for each actuator, the gradient decent algorithm must first gather a data set. To uphold the current level of abstraction, the physics engine's simulator is used to produce the data set by retrieving the linear and angular velocities for every possible combination of actuator states with each actuator using a state space of $\{-1, 0, 1\}$.

$$\left.\begin{array}{ccc|ccc} a_{0,0}, & \dots, & a_{n-1,0} & X_{0,0}, & \dots, & X_{5,0} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{0,3^n-1}, & \dots, & a_{n-1,3^n-1} & X_{0,3^n-1}, & \dots, & X_{5,3^n-1} \end{array}\right. \tag{10}$$

The resulting data set is depicted in Equation 10 where a represents actuator states, X represents the feature vectors and n represents the number of actuators in the system.

VI. Performance Evaluation

In order to evaluate the performance of our system, we implemented our tests on a Roomba device and compared the results with that of an equivalent program we wrote using the Roomba's own Open Interface API. For each test, the respective systems were programmed to preform a specific set of actions. The systems were limited to only using half of their maximum speed to capture the average expected performance.

The first set of experiments measures the accuracy of the odometry on the respective system when moving in a straight line. We placed the Roomba at a marked location and requested the Roomba to travel for a certain number of centimeters. After the program determined it had traveled that distance, it stops and the real traveled distance was measured which is then subtracted from the desired distance to get the offset. Tests were divided into increments of 10 cm where each test was run five times and the average was taken. Figure 2 shows the results of this test.

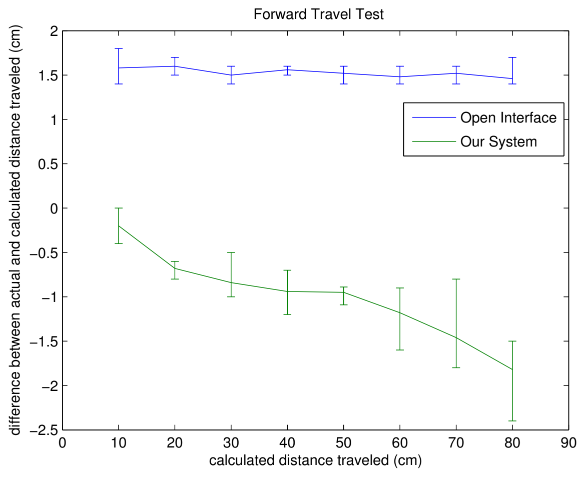

Fig. 2. Difference between odometry and actual traveled distance

As one can see, the Open Interface maintains a very steady offset where the Roomba traveled about 1.5 cm further than it was told to travel where as our system achieved more accurate results at shorter distances but the performance gradually declines. After about 60cm of traveled distance our accuracy is equivalent to the Open Interface. This relationship is emphasized in figure 3 where we took the results and divided them by the total distance traveled to get the relative offset.

In the next set of experiments, we compare the ability of the systems to accurately measure the speed of rotation. For each system, the Roomba is programmed to rotate in place for a specified number of complete rotations and measure the offset from the desired angle. For each number of rotations, we run the experiments five times and take the average. Figure 4 shows the offset obtained as a function of the rotation degrees.

As one can see from these results, both systems become less accurate the longer they are requested to rotate though our system maintains a higher accuracy. To analyze this result further, we divided the offset by the distance it has rotated. These results are displayed in Figure 5.

The two systems approach similar accuracies over time. Our system experiences a decrease in accuracy as it approaches an accuracy of 0.025 degrees off per degree while

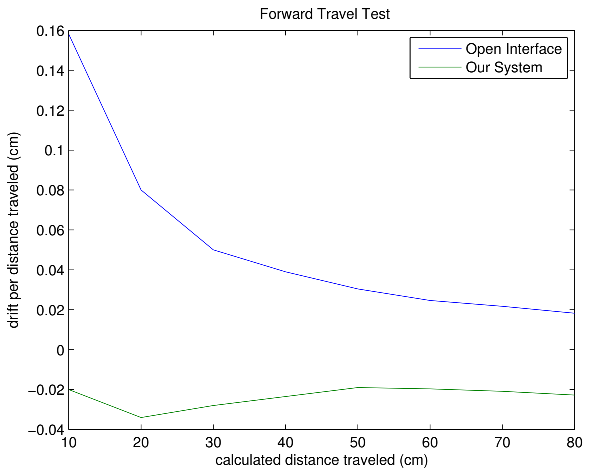

Fig. 3. Relative distance offset

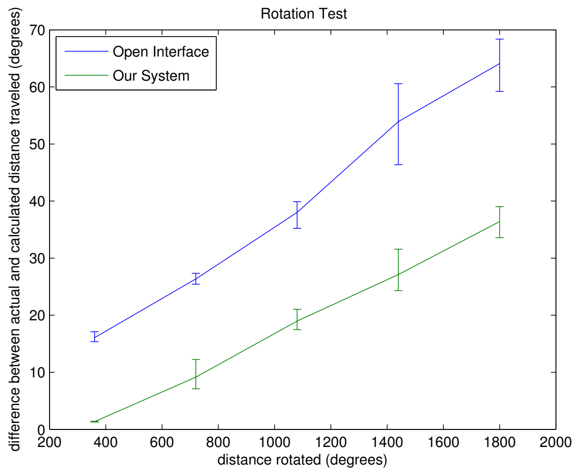

Fig. 4. Difference between odometry and actual rotated distance

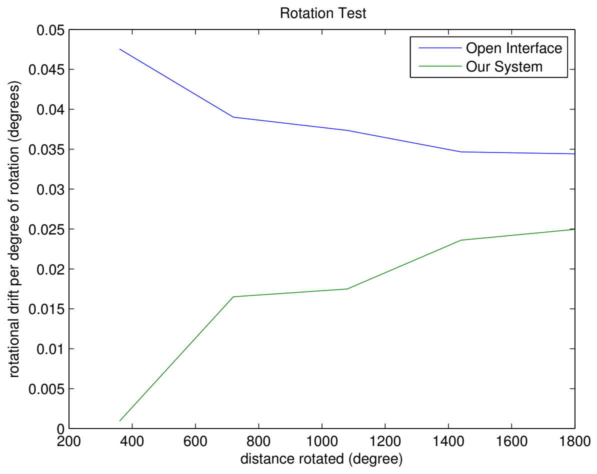

Fig. 5. Relative rotation offset

the Open Interface solution an improved accuracy as it approaches an accuracy of 0.034 degrees off per degree.

We also tested the ability of this control system to be implemented on different mobile devices. Unfortunately, due to resource restrictions we could not acquire separate hardware for this test. To conduct this test, we altered portions of the Roomba driver in order to emulate a different device. In particular, we limited the motor drivers by rejecting commands to move backwards thus changing the movement pattern of the Roomba. To test the proper handling of different hardware, we ran an identical program on both systems. The program commanded the Roomba to travel straight for 50cm, stop, turn right 90 degrees, and repeat until the Roomba has made a full loop. The notable change between these two system is that the Roomba with full movement can turn by pivoting around its center while the Roomba with limited drivers must take a wider turn by pivoting around one of its wheels.

We ran this program on both configurations five times and measured the distance that the final position differed from the initial starting position. The results of the two configurations were very similar. The unaltered driver produced an average inaccuracy of 0.98 cm with a max and min inaccuracy of 1.82 cm and 0.32 cm, respectively. The limited driver produced an average inaccuracy of 0.69 cm with a maximum of 1.06 cm and a minimum of 0.57 cm. The limited driver produced a more accurate average while the unaltered driver produced a smaller minimum inaccuracy.

VII. Conclusions

We demonstrated a method for controlling a MCPS at a high level abstracted from the hardware. The control system can remove the need to redesign and re-implement odometry calculations for new robotics hardware. While the calculations for Roombas and other similar hardware are quite simple, this system has the potential to be beneficial to more complex hardware that would normally require much more complex calculations. Since the calculations used by this system are the same that could be used by any other MCPS, only a physics model of the new MCPS would be required to take advantage of this system.

This system has also been proven to be more accurate than current methods in certain scenarios. All robotic systems can benefit from more accurate odometer calculations especially low-end hardware that is not equipped with high resolution sensors and other components to help calculate their positions.

While our current work is a great first step, there is more that could be done in future work. The physics engine requires a high degree of fine tuning to reach its current level of accuracy. This process could be partially automated given a set of test data and a proper set of machine learning algorithms. A proper automated tuning feature could potentially yield a greater accuracy than was demonstrated in this paper.

Acknowledgment

This research is funded in-part by NSF award #1149397.

References

[1] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative Multi-Robot Exploration," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2000.

[2] W. Haynes M. Zapata, N. Kannen, M. Sullivan, and J. Conrad, "An Autonomous Vehicle for Space Exploration," in *Proceedings of IEEE Southeastcon*, Huntsville, AL, 2008.

[3] T. Fong, C. Thorpe, and C. Baur, "Multi-robot Remote Driving with Collaborative Control," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 4, pp. 699–704, 2003.

[4] A. Marino, F. Caccavale, L. Parker, and G. Antonelli, "Fuzzy Behavioral Control for Multi-Robot Border Patrol," in *Proceedings of the 17th Mediterranean Conference on Control and Automation*, Thessaloniki, Greece, June 2009.

[5] A. Marino, L. Parker, G. Antonelli, F. Caccavale, and S. Chiaverini, "A Modular and Fault-Tolerant Approach to Multi-Robot Perimeter Patrol," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Guilin, China, December 2009.

[6] R. Murphy, "Rescue Robotics for Homeland Security," *Communications of the ACM*, vol. 47, no. 3, 2004.

[7] D. Stormont, "Autonomous Rescue Robot Swarms for First Responders," in *Proceedings of the Computational Intelligence for Homeland Security and Personal Safety*, Orlando, FL, April 2005.

[8] H. Okada, T. Iwamoto, and K. Shibuya, "Water-Rescue Robot Vehicle with Variably Configured Segmented Wheels," *Journal of Robotics and mechatronics*, vol. 18, no. 3, pp. 278, 2006.

[9] U. Zengin and A. Dogan, "Real-Time Target Tracking for Autonomous UAVs in Adversarial Environments: A Gradient Search Algorithm," in *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, CA, December 2004.

[10] J. Kim and Y. Kim, "Moving Ground Target Tracking in Dense Obstacle Areas Using UAVs," in *Proceedings of the 17th IFAC World Congress*, Seoul, South Korea, 2008.

[11] J. Borenstein, H.R. Everett, L. Feng, and D. Wehe, "Mobile Robot Positioning: Sensors and Techniques," *Journal of Robotic Systems*, vol. 14, April 1997.

[12] *Relative Position Estimation in a Group of Robots*, 2003.

[13] Shraga Shoval and Johann Borenstein, "Measuring The Relative Position And Orentation Between Tow Mobile Robots With Binaural Sonar," in *International Topical Meeting on Robotics and Remote Systems*, Seattle, Washington, March 2001.

[14] Fre'de'ric Rivard, Jonathan Bisson, Francois Michaud, and Dominic Le'tourneau, "Ultrasonic Relative Positioning for Multi-Robot Systems," in *IEEE International Conference on Robotics and Automation*. 2008, IEEE.

[15] John Kerr and Kevin Nickels, "Robot Operating Systems: Bridging the Gap between Human and Robot," in *44th Southeastern Symposium on System Theory*. 2012, IEEE.