Integration of Modern Robotic Software and AI Algorithms

MIKE LEWIS, CARNEGIE MELLON UNIVERSITY

Open-source software for robotic systems, while free and accessible, tends to suffer from the same problems afflicting other open-source software: weak documentation, steep learning curve for libraries, broken packages, dying support, etc. Moreover, most robot software focuses just on hardware control and intercommunication. Artificial Intelligence (AI) exists mainly in the form of algorithms that need to either be bolted onto robotic control software, or reimplemented to facilitate tighter integration. This article follows an exploration of the complexity in building a complete, intelligent robot from the ground up using the most popular software, hardware, and AI algorithms to autonomously solve the puzzle game, Rush Hour.

General Terms: Robotics, Articificial Intelligence, Integration, Planning

Additional Key Words and Phrases: Rush Hour, software, hardware, planning algorithm, OpenCV, BeagleBoard, ROS

INTRODUCTION

Many people comment on an apparent fact that the “robot revolution” should have hit by now. They claim that all the technology, both hardware and software, is here and often ask the question, “where are all the robots?”. The first part of my answer to this question is this: they are here, but just not in the places that the average person would be looking. No, you wouldn't find them rolling down the sidewalk, or walking through a plaza. Rather, they're in research labs, universities, high schools, and even elementary schools. The Arduino-servo method for building a robot is so easy and simple that young children can build them within minutes.

The second part of my answer to the question is that there is gap between the robots that are so easy to build and what the public expects robots to be. Light-sensing, differential-drive, Arduino-based robots are one thing. Robots that can take vocal commands, execute generic tasks, and autonomously navigate through a shopping mall are a completely different matter. This is mostly due to the fact that building a truly intelligent robot is an inherently difficult problem that apparently no one has been able to solve yet. And it's not a recent problem; researchers have been working on this problem for fifty years. The reason, I believe, for the slow progress is that there is very high barrier to entry for most people to contribute to this research. Thus, there are very few people working on the problem. Compare this with the speed of advancement of traditional computer science research. These fields have many more people working in them, using far more mature, refined technologies.

This work is supported by the Master of Science in Robotic Systems Development (MRSD) graduate program at Robotics Institute, Carnegie Mellon University.

Author’s address: M. Lewis, Robotics Institute, Carnegie Mellon University.

Fig. 1. Rush Hour by Thinkfun.

In the following sections, I will be documenting my experience with putting together an intelligent robot using the most popular software platforms, libraries, and packages and integrating it with an AI planning algorithm to solve the puzzle game, Rush Hour as seen in Figure 1.

1.1 Basic Definitions

The word “robot” used throughout this article refers to a system that accepts input through a sensor, manipulates the input data, then returns an output that is based on the data in the form of an action via an actuator.

The phrase “intelligent robot” refers to a robot as stated above, but with additional capacity in the traditional domains of Artificial Intelligence.

Rush Hour is a logic game by Thinkfun, Inc. The game consists of a 6x6 grid-style game board with coloured pieces in the shapes of cars and trucks. The pieces can only be moved in one degree of freedom depending on either a horizontal or vertical orientation. The goal is to move the pieces in a particular sequence, such that the red piece is free to move through the slot on one edge of the board. Rush Hour comes with a stack of cards that the player can use to set up various arrangements of the game pieces that vary in difficulty. The game is intended for ages 8 to Adult.

1.2 Problem Formulation

The objective of this research was to identify key hurdles for building an intelligent robotic system that integrates robotic capability with artificial intelligence to solve Rush Hour. The system was comprised of mostly off-the-shelf hardware and software components that are easily accessible, full-featured, and widely supported.

The main components of the system were a camera using computer vision to identify and encode the state of the game board, a planner that would determine the steps needed to solve the puzzle, and a manipulator to move the game pieces. The supporting components will be detailed in the following sections.

1.3 Hardware

The core of any computer hardware is the microprocessor, and for a robot it is no different. As was mentioned before, an Arduino-based platform simply does

not have the capacity for higher-level operation. In order to use the software tools that are necessary to tackle this task, such as Open Source Computer Vision (OpenCV), a 32-bit processor that is capable of running a common operating system is required. I chose the BeagleBoard by Texas Instruments, which is an open-hardware single-board computer that is cheap, widely available, and has a strong development community around it.

While the Kinect by Microsoft is possibly the most popular camera for low-budget robotics today, its form factor did not fit that of the robot I had in mind. Interestingly enough, another video game console peripheral was a better choice – the Playstation Eye (PSEye) by Sony. Though it doesn't come with built-in depth perception capability, a benefit was that it is much smaller. The PSEye is also supported by the Robot Operating System (ROS), which I will discuss later.

Mobility is a requirement for the robot that I came up with to solve the issue of game piece visibility and occlusion. The issue is that, when viewed from the side, taller game pieces in Rush Hour could hide others and make it difficult for the robot to accurately encode the game board state. Giving the robot the ability to move around and see the board from different angles would resolve this issue. It is worth noting that an alternative method is to simply make the robot taller.

The manipulation component of the robot was intended to be implemented with an arm made out of sub-micro-class servos. However, given the complexity of the system and the project already, this had to be scrapped in favor of giving more attention to other parts of the project. Nevertheless, the mobility and manipulation servos needed to be controlled somehow. Though I explored the possibility of having an Arduino handle the control based on commands received from the BeagleBoard, I decided it was simpler to purchase a 16-channel servo controller with an I^2C interface and connect it directly with the BeagleBoard's expansion port.

Power to the servos was supplied through the servo controller by a 6 Volt, 900 mAh Lithium Polymer Battery with a discharge rate of 20C. The BeagleBoard required a regulated 5 Volt supply, so it was powered separately with a wall brick. This did restrict its movement, but in practice it was not much of a problem.

I fabricated the chassis myself using High-Density Polyethylene (HDPE), which is durable type of plastic than is easy to machine. One thing that I did not know about HDPE is that it is particularly abrasive when reduced to a particulate form, such as when filed or sanded.

1.4 Software

The software platforms and libraries that I used were the most popular ones available. Robot Operating System (ROS) by Willow Garage is a messaging layer that can serve as a platform upon which to build a robotic control system. It uses a publish/subscribe and service-oriented communication style to facilitate communication between hardware and software nodes. OpenCV is

a computer vision library that is also maintained by Willow Garage. Both of these are best supported when run on Ubuntu Linux, which is what I decided to use for the robot's operating system. Had I not required these two software packages, I could have used any 32-bit operating system that can be built against the ARMv7 instruction set.

The first thing I had to do was get Ubuntu installed on the BeagleBoard. Two websites were very useful toward this [Sobral 2012] and . However, I encountered many issues across different versions of Ubuntu, different Linux kernel versions, and various pre-installed SD card images.

1.5 Planning

The initial plan was to develop a planner that would take in a Rush Hour game's initial state, compute a solution to the puzzle, then output the plan back to either the robot's manipulator or the operator.

ALGORITHM 1. Rush Hour domain in PDDL

```
(define (domain rush_hour)
     (:predicates
          (at ?car ?loc)
          (start ?car ?loc)
          (end ?car ?loc)
          (left ?loc_a ?loc_b)
          (above ?loc_a ?loc_b)
          (horiz ?car)
          (in_motion))

     (:action mv_up
          :parameters
               (?car ?from ?to ?from_bk ?to_bk)
          :precondition (and
               (at start (not (horizontal ?car)))
               (at start (top_of ?to ?from))
               (at start (top_of ?to_bk ?from_bk))
               (at start (start ?car ?from))
               (at start (end ?car ?from_bk))
               (at start (forall (?car) (not (at ?x ?
to))))
               (at start (not (in_motion))))

          :effect (and
               (at start (in_motion))
               (at end (not (in_motion)))
               (at end (not (start ?car ?from)))
               (at end (not (end ?car ?from_bk)))
               (at end (start ?car ?to))
               (at end (end ?car ?to_bk))
               (at end (not (at ?car ?from_bk)))
```

```
                    (at end (at ?car ?to)))
    )
)
```

Determining a representation for the Rush Hour domain and any of its planning problems was a critical first step. Planning Domain Definition Language (PDDL) is the most common way to represent planning problems, so this is what I went with. The best way that I came up with was to use the natural grid that makes up the game board and encode the positions of the pieces as single spaces that are linked together depending on the size of the game piece (either two or three spaces) and their orientation (horizontal or vertical). The predicates included relations between the pieces such as “above” or “left”. The resulting PDDL domain representation for Rush Hour can be seen in Algorithm 1.

Part of the plan was to quickly validate the domain representation with an existing planner. However, it was nearly impossible to find any partial-order planner that would compile and/or execute. The only working planner that I found was a STRIPS planner by Tansey, that would only accept ADL input. Thus, I was not able to test my domain representation and catch the issues that still remain a part of it.

Moreover, building a partial-order planner from scratch turned out to be more difficult than I had anticipated. With time in short supply, I was only able to implement several C++ classes and part of the algorithm.

CONCLUSIONS

Building an intelligent robot out of the most accessible, documented, and available components, hardware and software, is a very difficult task. There were numerous situations where my personal experience with Linux kernel development helped me solve some of the initial installation problems that I would expect a novice would have significant problems with.

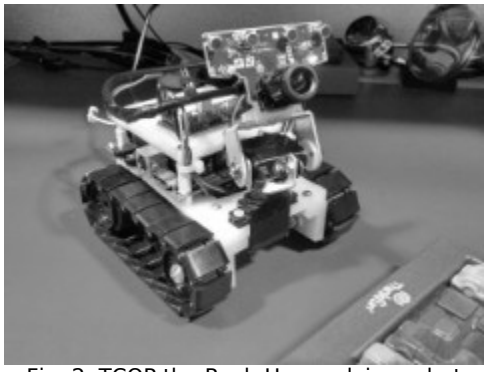

Fig. 2. TCOP the Rush Hour solving robot

The fact that the planning algorithm had to be re-implemented in the first place is unfortunate. I was unable to find any common AI libraries that could easily interface with the other components in my system.

By the end, a robot capable of object detection and discrimination based on blob colour was produced, but without the ability to solve Rush Hour.

ACKNOWLEDGMENTS

The author would like to thank Dr. Manuela Veloso, Dr. Tuomas Sandholm, John Dickerson, and Prateek Tandon of Carnegie Mellon University for providing education and guidance throughout the Graduate Artificial Intelligence course curriculum.

REFERENCES

Andrews Sobral. 2012. Embedded Computer Vision Platform with PandaBoard. Retrieved April 15, 2013 from http://sites.google.com/site/andrewsobral/pandaboard