Creating a Multi-Robot Stage Production

Junyun Tay[1,2], Somchaya Liemhetcharat[3] and Manuela Veloso[1]

[1]Carnegie Mellon University, USA
[2]Nanyang Technological University, Singapore
[3]Institute for Infocomm Research, Singapore
junyun@cmu.edu, liemhet-s@i2r.a-star.edu.sg, veloso@cmu.edu

Abstract. A multi-robot stage production is novel and challenging as different robots have to communicate and coordinate to produce a smooth performance. We made a multi-robot stage production possible using the NAO humanoid robots and the Lego Mindstorms NXT robots with a group of undergraduate women who had programming experience, but little experience with robots. The undergraduates from around the world were participating in a three day workshop – Opportunities for Undergraduate Research in Computer Science (OurCS), organized by the School of Computer Science from Carnegie Mellon University that provide opportunities for these undergraduates to work on computing-related research problems. They were given twelve and a half hours over a span of three days to familiarize themselves with the robots, plan the storyboard of the performance, program the robots, generate a multi-robot performance and create a presentation on what they learned and did. In this paper, we describe the tools and infrastructure we created to support the creation of a multi-robot stage production within the allocated time and explain how the time in the workshop was allocated to enable the undergraduates to complete the multi-robot stage production.

1 Introduction

A successful multi-robot stage production is extremely difficult, especially when different robot platforms are used and multiple robots have to communicate and coordinate with little human intervention to produce a coherent performance. Using multiple robot platforms requires familiarity of the robots' capabilities and a good understanding of how to program the robots. Manually controlling each robot individually requires a lot of practice between humans and is especially tedious when the robots have multiple degrees of freedom (DOF), e.g., the NAO humanoid robot has 21 DOF. Automating the behaviors of the robots, multi-robot communication and coordination are crucial to the synchronization of the robots' performance. In this paper, we explain the tools and infrastructure we created and describe our plan to support such an endeavor.

OurCS is organized to provide opportunities for undergraduate women to have a chance to work on computing-related problems and attend talks to learn about life in graduate school. We wanted to enable the participants to have a chance to work with multiple robots and different robot platforms within the short period of time that they were given. We conceived the idea of a multi-robot stage production, but no script was provided to the participants. The participants were allowed to exercise their creativity, yet at the same time, were limited by the capabilities of the robots. Hence, without a

strong understanding of the robot's hardware and software constraints, they would be unable to produce an entertaining multi-robot stage production. The participants were asked to sign up for the project beforehand and assigned to the research project based on available slots. At the end of the workshop, the participants had to produce a 5-8 minutes stage production and presentation to share what they learned.

We chose two robot platforms – the Lego Mindstorms NXTs and the NAO humanoid robots (Fig. 1). The Lego Mindstorms NXTs are commercially available in stores, but not the NAO humanoid robots as they are much more expensive. The NAO humanoid robots are not configurable, unlike the Lego Mindstorm NXTs, but are more complex in the number of DOF. To enable the Lego Mindstorms NXTs to communicate one another, we used the XBee radios, which is not packaged with the Lego Mindstorms NXT. Having these two robot platforms allow the participants to have a unique experience that is difficult to get elsewhere. We also aimed to ensure that every participant has a chance to work on each robot platform and learn about the individual hardware and software capabilities. The Lego Mindstorms NXTs were already constructed with a arm that can rotate, wheels to move around and XBee radios to communicate with one another. Extra software functions were written in RobotC to enable the NXTs to send robot to robot messages. For the NAO humanoid robot, external software functions were written to allow motions to be easily synchronized with speech, music and LEDs in the eyes of the NAO humanoid robot. We planned our tools and infrastructure such that the participants were able to use these software functions to create a performance that enable the robots to communicate, synchronize the motions of the robots with text-to-speech or music and change the LEDs of the NAO robots.

All the participants had programming experience, but only some had experience with the Lego Mindstorms NXTs. None of them had worked with the NAO humanoid robot. At the end of the workshop, all of them had a chance to work on all the robots and put together an entertaining and informative eight-minute presentation to explain what they learned and did. We successfully enabled a group of undergraduates with little experiences with robots to create a multi-robot stage production in twelve and a half hours using two NAO humanoid robots and three Lego Mindstorms NXT robots. It is rare to see opportunities for participants to experience working on multiple robots, let alone different robotic platforms, specifically Lego Mindstorms NXTs and NAO humanoid robots in our case. Careful planning is needed to allow the participants to complete a multi-robot stage production within a short period of time and to gain useful insights about multi-robot coordination and communication, lessons that can never be learned with a single robot.

In Section 2, we review what others have done in the area of educational robotics. Next, in Section 3, we describe the preparations we made to facilitate learning of multiple robot platforms, multi-robot communication and coordination within a short period of time. We explain the rationale and the inner workings behind the tools and infrastructure created so that the participants can focus on creating the motions and behaviors of the robots based on the storyboard they planned and use abstractions of robot functions without in-depth knowledge of the code for multi-robot communication and coordination. Following that, in Section 4, we list the learning objectives and the activities that we planned out for the participants to enable them to effectively learn about the

(a) NAO humanoid robot: 21 rotational joints; LEDs in the eyes; speakers in the ears; text-to-speech.

(b) Lego Mindstorms NXT robot: Two motors for the wheels; one motor for the arm; XBee communication module; internal speaker.

Fig. 1: Robots and their features. Images are not to scale.

hardware and software of the robots. In Section 5, we recount the actual day-to-day activities of the participants. Lastly, we outline what the participants learned and discuss our insights from the planning to the implementation and execution of the multi-robot stage production in Section 6.

2 Related Work

Carnegie Mellon Robotics Academy developed materials to teach multi-robot communications [3], using robot platforms such as LEGO Mindstorms NXT, the VEX robotics kit and the Arduino family of processors and ROBOTC. ROBOTC was used as the programming language as ROBOTC can be used for these robotic platforms with little to no changes in the code. Avanzato discussed the use of low-cost educational robot platforms and high-level software support for existing educational robot platforms based on the materials from Carnegie Mellon Robotics Academy [1]. Avanzato also described a "multi-robot design challenge for a regional robot contest, multi-robot classroom and laboratory activities, and a programmable controller for multi-robot communication are presented" [1]. Specifically, four autonomous mobile robots, comprising one NXT and three VEX robots were designed and implemented to cooperatively explore a maze with the goal of extinguishing multiple candles that were randomly located [1].

Lego Mindstorms has always been used as a educational tool to teach robotics due to its low cost and simplicity. Casini et al. have developed a remote laboratory for multi-robot systems using LEGO Mindstorms NXT technology [4]. Remote users can design control laws in Matlab and test them by performing experiments remotely using the team of robots available in the experimental setup. There is a global vision system, which "simulates different types of sensors and communication architectures" [4]. Users will only be able to remotely control the NXTs robots but not interact with the robots directly. Others used software such as LabView [6]. Franklin and Parker proposed "Overwatch as an inexpensive educational tool for teaching and experimenting in multi-robot systems" [5] and used Scribblers as the robot platform to experiment

with a multi-robot system [5]. McLurkin et al. proposed using Rice r-one mobile robots for multi-robot curriculum and described the courses that they implemented using their platform [8] to teach multi-robot concepts. Others used e-pucks which are designed for education in engineering [9]. Many robotic platforms and software have been developed for multi-robot education. All these robot platforms are mostly low cost and less complex than humanoid robots. By using the Lego Mindstorms NXT and the NAO humanoid robots, we allow the participants to have a unique experience with two very different robot platforms in terms of hardware and software.

3 Preparations

In this section, we describe the tools and infrastructure we prepared before the workshop to support the multi-robot stage production. The tools and infrastructure were built upon previous work and described in Section 3.1 for the NAO humanoid robots and in Section 3.2 for the Lego Mindstorms NXT robots.

3.1 NAO Humanoid Robots

Our RoboCup team, CMurfs (Carnegie Mellon United Robots For Soccer) [7], participated in the RoboCup Standard Platform League competition using the NAO humanoid robots to play soccer. We show the Cognitive Agent, a part of our RoboCup code architecture, which processes the Robot State and generates a Robot Command. A Robot Command comprises a Motion Command (e.g., walk, perform a motion), a Speech Command (to be used by the text-to-speech engine), and an LED command (to display colors on the robot). More details of our code architecture can be found in [7]. The Cognitive Agent consists of various components, namely:

- Agent Manager is a component that passes the information between components.
- Game State Manager is used to update the game state based on the messages received from the Game Controller, as well as button presses on the NAO robot. The Game Controller is used in a game in the Standard Platform League of RoboCup to synchronize the game state across all NAO robots in the game. The game state consists of states such as the start of the game, penalty shot etc.
- Vision receives the camera images and analyzes the images to determine features such as ball, lines, goal posts, robots etc. that are important in the game.
- Localization determines the global position and orientation of the robot in the game using features from Vision and odometry based on motion commands.
- World Model keeps track of the robot's hypotheses of the ball and the information shared by the other robots (teammates).
- Behaviors generate robot commands based on the features and information available and also determine the messages to send to teammates. In order to make decisions, Behaviors is built on a Finite State Machine (FSM) model, which is described in detail in Section 4.2. A FSM is made up of several states and the transitions between states are based on the conditions programmed.
- Log allows messages to be sent across the network to a Remote Display Client. This component is useful for debugging the states and information available on all the robots.

– Communications is in charge of sending and receiving messages between robots and a Remote Display Client.

Each component has their own input and output interfaces and allows us to easily configure the use of each component. We can easily enable and disable components based on the needs. This flexible and configurable architecture enables us to use the code from our RoboCup team as the functionalities we require to support the a multi-robot stage production exist. We disable the use of the Game Stage Manager, Vision, Localization, World Model and Log in the workshop. The rest of the components enable us to achieve the following: (1) communication between the NAO robots; (2) ability to execute motions; (3) set the LEDs in the eyes; (4) text-to-speech capabilities.

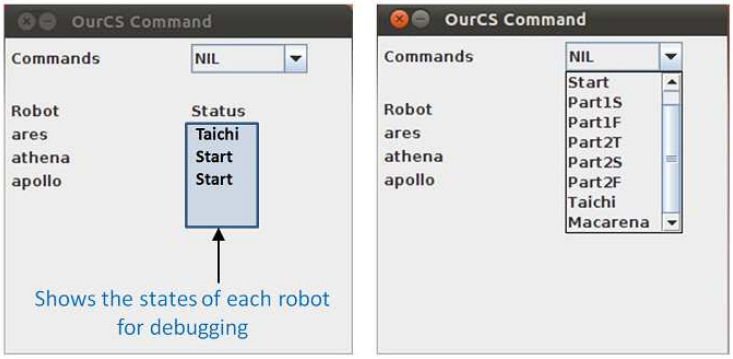

Fig. 2: Snapshot of the Puppet Master UI (left) and the list of commands sent (right).

We replaced the Game Manager with a new component, called the Puppet Master. We created a "puppet master" program, *OurCS Command*, shown in Fig. 2, where the participants could use the computer to send messages to the robots to start or stop behaviors of the robots via a user interface shown in Fig. 2. The user interface shown in Fig. 2 reads in a list of commands from a text file before it starts up. These commands can be selected and sent to all the robots as messages. The robots that can receive the messages are listed in the list of robots shown in Fig. 2. The list of robots and the IP addresses of the robots can also be configured via another text file. The participants can also debug the states of the FSM running in the robots. We used UDP communication between *OurCS Command* and the NAOs, but the message protocol was abstracted from the students. In this way, the students focused on creating messages and how the messages influenced the FSM transitions between states, without worrying about how the messages were transmitted from the computer to the robot. We also added specific robot-robot and computer-robot messages so that the NAOs and the Puppet Master can communicate the states in the FSM.

We also created a component that automatically exports motions from Choregraphe into a format compatible with our RoboCup code. Choregraphe is a piece of software created by Aldebaran Robotics, the manufacturer of the NAO humanoid robot, that allows users to easily generate motions on the NAO. We will explain in detail how Choregraphe is used in Section 4.1.

To enable the participants to program the NAO to execute motions, play a wave file, use the text-to-speech capabilities, and change the LED colors in the eyes simultaneously, we created a function wrapper: MacroAction(Motion, TextToSay, WaveFileName, LeftEyeColor, RightEyeColor). In this way, the students could focus on creating behaviors (sequences of macro actions) for the performance, without having to handle synchronizing different types of actions or multiple threads and processes on the NAO. The students can use the function wrapper instead. of sending 4 separate commands.

Algorithm 1 Snippet of sample code showing a FSM and MacroAction in action

```
1:  if fsm.inState(Part1F) then
2:      if fsm.isNewState() then
3:          // assumes that the actionListStatus = UnLoaded
4:          actionList1F.clear()
5:          MacroAction tempAction(StaticAction::actShakeHead, "Terrible moves, N X T", "",
            0xFF0000, 0xFF0000, Action::BothEyes)
6:          if MY_ROBOT_NAME == ROBOT1 then
7:              actionList1F.push_back(tempAction)
8:          end if// Thse two lines must be included to load MacroActions
9:          currentActionList = &actionList1F
10:         actionListStatus = Loaded
11:     else
12:         if actionListStatus==UnLoaded then
13:             // loaded the action list and executing
14:             if commandToExecute=="Part2T" then
15:                 // commandToExecute is the command sent by Puppet Master
16:                 fsm.trans(Part2T, "Done with Part1F")
17:                 continue
18:             end if
19:         end if
20:     end if
21: end if
```

We prepared two laptops to be used with two NAO robots respectively. We installed NaoQi, Choregraphe and our RoboCup code on the laptops. We also prepared sample code to illustrate how the FSM and MacroAction classes can be used. The participants can refer to these code or edit the code to suit their needs. The sample code is shown in Algorithm 1.

3.2 Lego Mindstorms NXT Robots

We previously created curriculum for students at the K-12 level to learn about multi-robot concepts using a variety of robot platforms such as the Lego Mindstorms NXT. We used ROBOTC, a programming language that can be compiled and downloaded to different platforms with little or no changes. Therefore, for the multi-robot stage performance, the participants could code in RobotC to actuate the robot and to pass messages from one robot to another. We built upon their experiences and curriculum developed for multi-robot communication and synchronization of motions. The lessons that we will use are described in detail later. We built two Lego Mindstorms NXT with three wheels where two are actuated and an actuated arm which can rotate.

4 Planning of day-to-day activities

In this section, we describe the day-to-day activities that we planned for the participants. We formed three groups of students where two groups work on the NAO robots and one group works on the NXT robots. The groups are formed based on the participants' interest. Each group is also led by a mentor, who is proficient in the robot platform and software. We plan to spend about four hours per day with the participants. On Day 2, we swap the groups so that everyone has a chance to work on the NAO and NXT robots.

4.1 Day 1

At the start of the day, we gave an introduction of the task – a multi-robot stage production. We gave a general overview of the hardware available on the NAOs and NXTs. We explained that the robots are limited in terms of perception because we disabled the use of sensors such as cameras, ultrasound sensors etc on the NAOs, except touch sensors on the NXTs. The use of cameras and vision are too complex and do not help the participants to learn about multi-robot communication and coordination.

Next, we explained the use of actuators and DOF of a robot. We highlight that the NAO robot is more complex in terms of the number of DOF where the NAO has 21 DOF whereas the NXT has 3 DOF. The NAO robot also has more sensors such as cameras, ultrasound, gyroscopes and accelerometer, touch sensors such as chest buttons and foot bumpers, LEDs in the eyes and text-to-speech capabilities. Though the Mindstorm NXT set includes many sensors such as light sensors, ultrasound sensors, we only added touch sensors to the NXTs that we built.

Following that, we break the participants into three groups, so that two groups will work on creating motions on the NAO robot and one group will work on creating motions on the NXT. At the end of the day, the participants should create a basic outline of the stage production. We described what we teach the participants in detail.

Choregraphe - Creating Motions on a single NAO robot To generate a motion on the NAO, we introduced the concept of a keyframe (static pose) and interpolation time shown as a timeline to create animated motions on the NAO (Figure 3). A motion is made up of several keyframes and timings to interpolate between keyframes are also defined. We taught the students how to use Choregraphe, a software to create motions easily provided by Aldebaran Robotics, the manufacturer of the NAO. We explained that robots have angular joint limits and maximum joint angular velocities that cannot be exceeded. The angular joint limits are shown in Choregraphe with sliders for each joint and the end of each slide of the slider shows the minimum and maximum joint angle. We also taught the participants how to record each static pose as a keyframe and create a sequence of motions.

Stability of a humanoid robot is extremely important as compared to a wheeled robot, especially if the NAO humanoid falls, since the cost of repairing the NAO humanoid robot is high. Therefore, we emphasized the balance of the NAO humanoid robot. We attached a harness to the NAO humanoid robot when the participants create motions so that they can hold on to the robot and ensure that it doesn't fall.

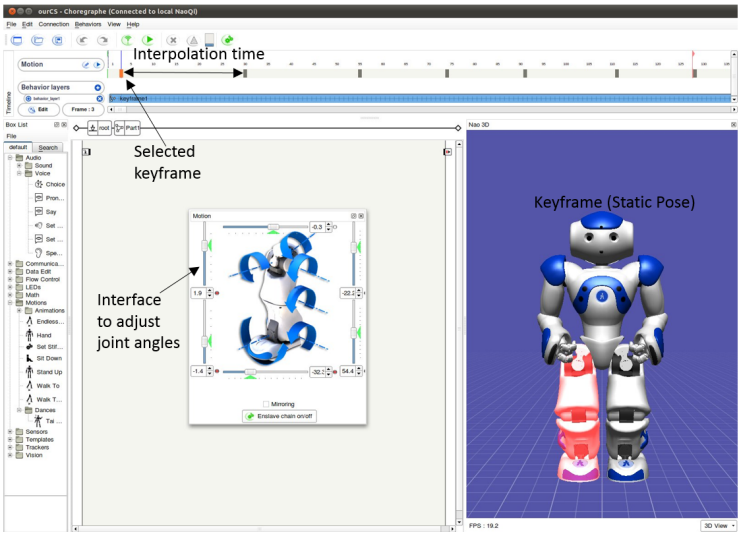

Fig. 3: Choregraphe

Commands to actuate a Lego Mindstorm NXT We first taught how motions on the NXTs are generated by actuating each motor independently at different speeds, and provided sample code of basic motions. The students then created new motions such as moving in an arc, and spinning the NXT's "arm".

We then introduced how the NXTs could communicate using the XBee radios. We explained the concept of using a common language, so that the robots understood messages that were sent and received. The students implemented a sequence of actions on two NXTs, where they sent messages to each other and took turns executing actions. Next, the students created a follow-the-leader sequence, where one robot would select a random action, inform the other robot, and both robots would execute the same action together. Thus, the students successfully understood the basics of multi-robot communication and coordination.

To synchronize the motions on the NXTs with the storyboard of the multi-robot stage production, the students also used a "puppet master" to switch action sequences within the NXTs. Within each sequence of actions, the NXTs communicated to take turns performing actions, and synchronize to perform identical actions together.

4.2 Day 2

For Day 2, the participants learn how to convert the motions of the NAO robot to code. They will also learn about Finite State Machine and to use the function wrapper, MacroAction. The participants also learn about multi-robot communication, specifically how to pass messages between the NAOs and the Puppet Master and how to pass messages between the NXTs. We did not prepare code to send messages between NAOs and NXTs as we believe that the participants will be able to pick up multi-robot communication concepts from what we already prepared.

Finite State Machine A finite state machine (FSM) enables them to implement their storyboard: (1) A FSM has a finite number of states and the story can be broken down into sequential steps; (2) The transition from one state to another is initiated by a triggering event or condition, e.g., a message received by the NAO from the "puppet master" (explained below), or the end of a sequence of motions. We also created code templates and sample code to provide examples on how to use our code, so that they could easily create behaviors for the NAOs without worrying about the code syntax. Although we provided the functionality of enabling message-passing between robots, the undergraduates did not use them as they decided to use the "puppet master" program to coordinate the robots. The participants had to code in C++, but were provided code templates as examples to follow.

Based on the curriculum and code developed for multi-robot communication, we list the lessons from [2] and summarized the multi-robot communication lessons:

1. Message passing: To communicate, the robot must be able to send and receive a message that comprise a string of limited number of characters. Relaying messages from one robot to another is not as simple as it looks. It requires the robot to send a message repeatedly to ensure that the message is received using the function void SendStringRepeated. When a message is sent as a string, the robot must be able to check if a message exists and read the message.
2. Guaranteed message delivery: To ensure that a message is sent and received, we have a function called SendStringConsistently where a message is guaranteed to be received by the recipients. We have another function SendStringConsistentlyTo where the message is guaranteed to be received by a particular robot.
3. Condition to initiate a motion: In multi-robot coordination, it is common for one robot to wait for a message to be received from another before a motion is initiated.
4. Map a message string to an action: After the participants learn how to actuate the robot, we teach the participants about mapping a string to an action. The string can be sent from one robot to another as a message and once the string is received, the robot can be actuated to perform an action (motion). This lesson is built on the previous three lessons.
5. Parameterized message string to action: Instead of defining fixed actions (motions) to be performed, we can parameterize the action. For example, for the robot to rotate by a certain number of degrees, we can pass the parameter 45 in the message so that the robot will rotate by 45 degrees.

4.3 Day 3

For Day 3, we plan to allow time for the participants to finish working on their stage production and do a full rehearsal before their presentation. We emphasize the importance of practice at the venue as behaviors of the robots may differ at the venue due to different conditions such as texture of carpet, causing variations in the motions. Also, the participants will have to practice their presentations and to synchronize the actions between the NAOs and the NXTs since there was no message passing between the NAOs and the NXTs. The NAOs can be synchronized via commands sent from the Puppet Master and the NXTs can be synchronized via commands sent from a computer or messages sent from another NXT.

5 Actual day-to-day activities

We describe the day-to-day activities that actually happened during the workshop. The participants only spent a few hours each day working on the robots as they had other talks and activities scheduled. We divided them into three groups and each group of participants is led by a mentor, who is familiar with the robot platform and software. Each group of participants only has 2-3 undergraduates. The participants had no script to follow except that the task was to produce a multi-robot stage performance. Hence, the participants had to learn, plan and execute a multi-robot performance using the NXTs and the NAOs and summarize what they learned in a 5-8 minutes presentation.

5.1 Day 1

On the first day, the participants spent four hours and forty-five minutes working on the robots. They familiarized themselves with the NAOs and NXTs and created motions on the robots. Lastly, the participants discussed and designed an outline of the multi-robot stage production. They found the music for the production and created motions for the stage production. They came up with the idea of a NAO tai-chi master teaching another NAO and two NXTs tai-chi. The robots then ended with the Macarena dance.

5.2 Day 2

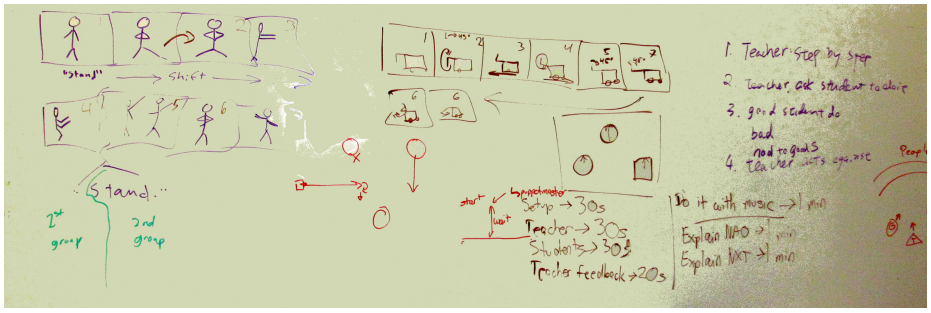

Fig. 4: Snapshot of Script

On the second day, the participants spent four hours working on the robots. The participants created a detailed outline and wrote them on the whiteboard (Fig. 4). Later, the participants refined the timeline to determine the timings of each event in the stage production and is listed here:

1. Introduction (30 seconds)
2. Tai chi master / student practice (90 seconds)
3. Tai chi demonstration by the two NAOs and the Lego NXTs (60 seconds)
4. Explanation of the NAOs (60 seconds)

5. Explanation of the NXTs (60 seconds)
6. Macarena (60 seconds)

After the participants planned the storyboard, they assigned responsibilities to each team member, such as the sections to present in the PowerPoint presentation. Following that, we taught them the concept of a finite state machine and multi-robot communication concepts such as message passing from one robot to another. The participants created motions for the Macarena dance and learned to incorporate the keyframe motions into the code. They learned how to use the function wrapper MacroAction and how to check for messages from the Puppet Master. The participants also swapped groups to learn about another robot platform that they did not learn on Day 1.

At the end of the day, the participants showed a demonstration of the motions created for the NAOs and the NXTs. However, the NAO-to-NXT communications had not been implemented, hence they sent commands to each robot type independently.

5.3 Day 3

On the third day, the participants spent three hours and forty-five minutes working on the robots. The participants finished the Tai Chi demonstration and added the Macarena dance into code. They also produced a PowerPoint presentation explaining what they learned. They also had to practice synchronizing the commands sent from the NAO and NXT Puppet Masters so that the robots appear synchronized in their performance. A summary of their work during the three day workshop, multi-robot stage production and presentation can be viewed at `https://youtu.be/0y9cG9lnKOk`.

6 Conclusion

The participants learned a lot from the workshop, which included all the concepts we wanted to impart in terms of multi-robot communication and coordination. They also learned a new model – Finite State Machine – and found them easy to apply in the workshop to initiate different states based on the messages sent. They also learned that creating stable motions for a humanoid robot is extremely difficult. Walking and moving legs apart such as sliding the legs across the floor is intuitive and easy for humans, but difficult to achieve using keyframe motions. They also learned that sliding legs apart for a humanoid robot on different carpet textures is difficult, hence they created a motion that lifts one leg up slightly before putting the leg down to create a sliding leg motion. They also learned that rehearsals are important as the behaviors of the robot may differ in different environments. They had to edit the motions of the robot due to the differences in carpet textures at the presentation venue. The feedback from the participants were that they learned a lot from these experiences with multiple robots and some wanted to continue pursuing graduates studies in robotics. One of the participants even spent a summer with us working on RoboCup.

In this paper, we addressed the challenges of enabling undergraduates with little robotic experience to create a multi-robot stage production in twelve and a half hours.

We described how the time was structured and how concepts and the software infrastructure were abstracted so that the students focused on the stage production. We divided the concepts we wanted to teach into manageable sizes, and students can apply what they learned immediately. The concepts were easy to understand based on their prior programming experiences. The students could also be creative in generating a script for the performance which keeps them motivated as they were involved from the planning to the execution of their plan. Hence, by sharing our experiences and how we prepare for this workshop to support a multi-robot stage production in a short amount of time, we hope that others can learn from our experiences.

Acknowledgments

We thank Brian Coltin for his guidance of the students who participated in the workshop. Junyun Tay is part of the NTU-CMU Dual PhD Programme in Engineering (Robotics). The views and conclusions contained herein are those of the authors only.

References

1. Avanzato, R.L.: Multi-robot communication for education and research. In: 2013 ASEE Annual Conference. ASEE Conferences (2013), https://peer.asee.org/22304
2. Carnegie Mellon Robotics Academy: FIRE Wiki, `http://www.robotc.net/firewiki/index.php/Main_Page`
3. Carnegie Mellon Robotics Academy: Multi-Robot Communications, `http://www.cs2n.org/activities/multi-robot-communications`
4. Casini, M., Garulli, A., Giannitrapani, A., Vicino, A.: A lego mindstorms multi-robot setup in the automatic control telelab. In: In Proceedings of 18th IFAC World Congress. p. 2 (2011)
5. Franklin, D.M., Parker, L.E.: Overwatch: An educational testbed for multi-robot experimentation. 26th International Florida Artificial Intelligence Research Society Conference (FLAIRS) (2013)
6. de Gabriel, J.M.G., Mandow, A., Fernndez-Lozano, J., Garca-Cerezo, A.: Using lego nxt mobile robots with labview for undergraduate courses on mechatronics. IEEE Trans. Education 54(1), 41–47 (2011)
7. Liemhetcharat, S., Coltin, B., Tay, J., Veloso, M.: CMurfs 2011 Team Description Paper. In: Proc. RoboCup 2011 CD (2011)
8. McLurkin, J., Rykowski, J., John, M., Kaseman, Q., Lynch, A.: Using multi-robot systems for engineering education: Teaching and outreach with large numbers of an advanced, low-cost robot. Education, IEEE Transactions on 56(1), 24–33 (Feb 2013)
9. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., christophe Zufferey, J., Floreano, D., Martinoli, A.: The e-puck, a robot designed for education in engineering. In: In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions. pp. 59–65 (2009)