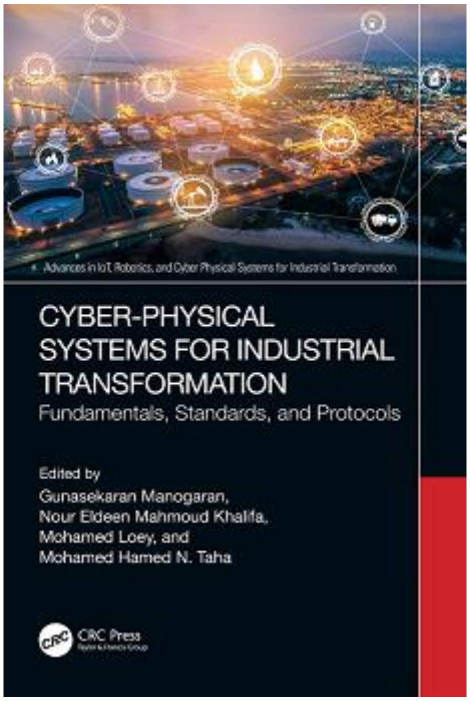
*Development of autonomous cyber-physical systems using intelligent agents and Lego technology*

Burak Karaduman, Geylani Kardas, Moharram Challenger

DOI: 10.1201/9781003262527-11

To appear in: *Cyber-Physical Systems for Industrial Transformation: Fundamentals, Standards, and Protocols (1st Edition)*

Published online: 5 April 2023



Advances in IoT, Robotics, and Cyber Physical Systems for Industrial Transformation

CYBER-PHYSICAL SYSTEMS FOR INDUSTRIAL TRANSFORMATION

Fundamentals, Standards, and Protocols

Edited by
Gunasekaran Manogaran,
Nour Eldeen Mahmoud Khalifa,
Mohamed Loey, and
Mohamed Hamed N. Taha

CRC Press

Please cite this article as: Burak Karaduman, Geylani Kardas, Moharram Challenger, Development of autonomous cyber-physical systems using intelligent agents and Lego technology, Cyber-Physical Systems for Industrial Transformation: Fundamentals, Standards, and Protocols (1st Edition), Manogaran et al. (Eds.), doi: 10.1201/9781003262527-11

# Development of Autonomous Cyber-physical Systems using Intelligent Agents and LEGO Technology

Burak Karaduman (Department of Computer Science, University of Antwerp, and Flanders Make, Belgium, burak.karaduman@uantwerpen.be) (0000−0002−7262−992X)

Geylani Kardas (International Computer Institute, Ege University, Izmir, Turkey. geylani.kardas@ege.edu.tr) (0000−0001−6975−305X)

Moharram Challenger (Department of Computer Science, University of Antwerp, and Flanders Make, Belgium, moharram.challenger@uantwerpen.be) (0000−0002−5436−607

## Abstract

Cyber-physical systems (CPS) have attracted various embedded technologies and researchers from low-level where the practitioners implement their systems using high-level programming languages where the multiple paradigms also overlap. As CPSs merge with numerous disciplines, heterogeneity emerges, and increasing complexity requires abstractions to program CPSs. Moreover, it is feasible to benefit from the suitable technologies that facilitate programming the physical parts of CPS. LEGO might prefer creating concrete use cases as composable technology, while embedded technology allows running the software to establish CPS. However, workflow, architecture, design alternatives, and abstraction should be defined to achieve this combination. Once low-level control is merged with agent-based programming, then this infrastructure can pave the way for applying intelligent-based solutions to tackle the high-level problems of CPS. This chapter introduces the architecture, a development workflow, and a set of agent-based CPSs to describe how LEGO technology-based CPS can be developed where software agents are integrated into the design, conforming to the provided architecture.

## 9.1 Introduction

The rise and advancement of networked systems have produced new paradigms and design challenges in embedded systems. The information processing and computation are merged with communication and control that creates Cyber-physical Systems (CPS) (Baheti and Hill, 2011). This evolution expands the capabilities of embedded technology interacting with the physical world through computation, networked communication, and control and paves the way for the cyber and physical future. During the interaction with the physical world, there is a phenomenon that has to be responded to by the system. The cyber part motivates the physical component of the system to change its state. Then, physical action creates a change in the environment, resulting in an event being maintained by the cyber part. This way, medical devices, vehicles, intelligent highways, robotic systems, and factory automation can be implemented, considering new capabilities achieved by CPS and integral paradigms using the multi-paradigm approach (Carreira et al., 2020).

The component variety of the CPS makes the system heterogeneous. These components should be controlled to react to environmental changes. In this way, the system can sustain during run-time, so software agents can be a way to program these complex systems.

In this chapter, we contribute to constructing the CPSs by providing example agent-based CPS implementations using LEGO technology. The examples are both from mobile and stationary systems. This way, the CPS's cyber and physical sides are addressed based on a proposed architecture. In addition, we provide a detailed workflow and implementation steps to provide better insights for the practitioners and researchers.

The rest of this chapter is organized as follows: Section 9.2 introduces multi-agent CPS and their relation. Section 9.3 gives a brief background of the BDI (Belief-Desire-Intention) agents, LEGO technology and involved hardware. Agent development frameworks which were used to implement the examples are presented in section 9.4. The architecture is proposed in section 9.5. Section 9.6 presents the development workflow of the examples. Concrete implementation for the agent-based CPS examples is given in section 9.7. Software excerpts related to the example agent-based CPS are shown and explained in section 9.8. The chapter is discussed, and technical notes are shared in section 9.9.

## 9.2 Multi-Agent CPS

Multi-agent paradigm implies autonomous software capable of acting dynamically, reactively and intelligently that alters its environment to create a state change based on defined behavioural features. Multi-Agent Systems (MAS) represent multiple autonomous agents collaborating to achieve individual tasks to reach a global goal. Agents collaborate to perform global tasks to enable solving problems in a joint effort that would be impossible to solve by a single agent. Software agents are deployed into the cyber part of the CPS. Generally, for a MAS, an information model of the physical world emerges from agents' mental states. Some studies in the literature propose to benefit from the multi-agent systems (Leitao et al., 2016; Sakurada et al., 2019; Karaduman et al., 2021; Karnouskos et al., 2020) as it is a suitable paradigm for providing smartness, decentralization, autonomy, and communication between subsystems and systems. They increase the effectiveness of a CPS by augmenting the target system with its functionalities. The software agents can (re)-configure the control parameters while monitoring the transition between tasks and observing human errors. They can enhance requirements such as product quality, in-time delivery, and efficient area exploration.

When agents obtain control over the components of the CPS, the practitioners can focus on higher-level approaches. Therefore, an integration of MAS and CPS may facilitate the programming of CPS applications such as Wireless Sensor Networks (Arslan et al., 2017; Asici et al., 2019) and the Internet of Things (Türk and Challenger, 2018). However, its physical conditions do not always allow us to directly implement these high-level approaches, considering the operational systems and dangerous environment. Furthermore, creating a skeleton system of an actual CPS may be a burden. However, it should sustain its functions, processes, and goals. Therefore, the target system can be miniaturized using a composable solution such as LEGO.

## 9.3 Background

This section gives background information about BDI Agents, Embedded Technology, LEGO technology and related embedded boards.

**BDI Agents:** The belief, desire and intention (BDI) model is a software reasoning approach that has been developed for programming intelligent agents. The BDI agents can balance the time spent in the deliberation phase by choosing what to do and executing the suitable plans as action(s). Beliefs are the information that belongs to the agent, other agents, and agents' surroundings. Desires represent all goals that can be potentially success-able states. Lastly, intentions are defined as any state of activities that were decided to realize.

**Embedded Technology:** Embedded technology allows programming a microprocessor, which is computing hardware, using embedded software. An application can be developed for performing any dedicated task. Embedded hardware binds the cyber world with the physical world using I/O ports.

**LEGO Technology:** Integrating MAS and CPS may enable high-level programming in various applications. Agents who control the CPS's physical components can help solve cyber problems using their reasoning mechanisms. However, it is not always possible to create an actual CPS because of cost, safety reasons, and the planned system's size. Therefore, the system should be scaled down while sustaining its functionality, accuracy, and goals. A compose-able and easy-to-construct technology, such as LEGO, represented in Figure 9.1, may help to miniaturize the existing system.

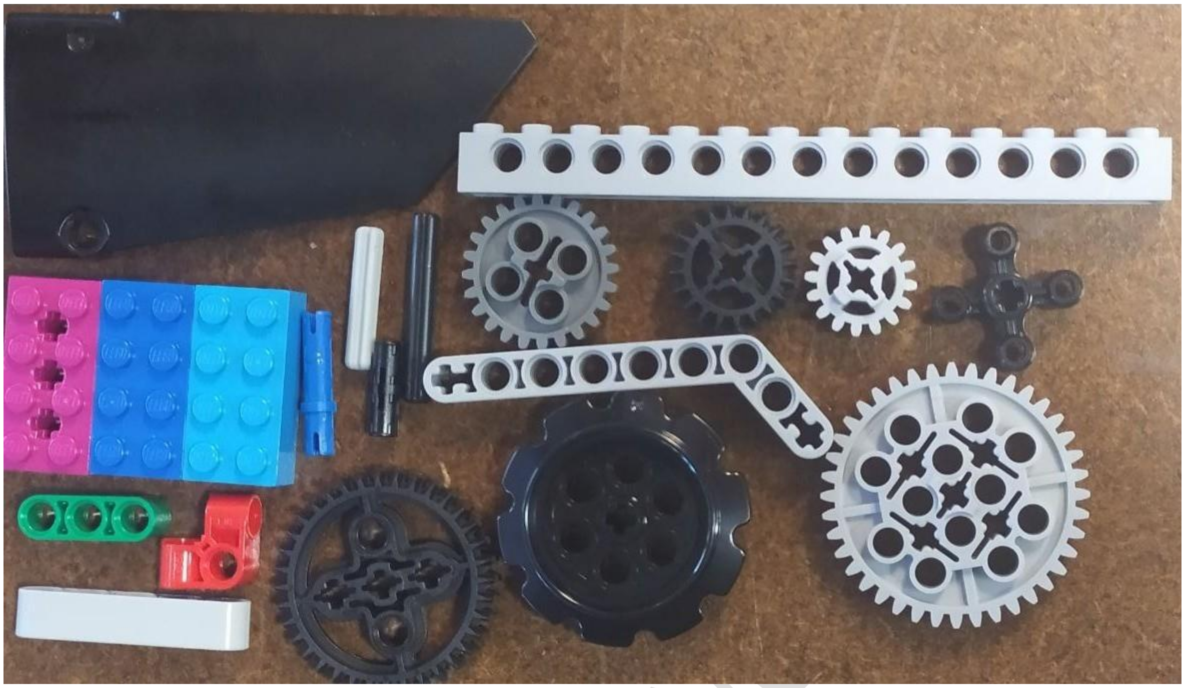


Figure. 9.1: Sample LEGO components.

**EV3 Hardware:** As represented by the left top side of Figure 9.2, EV3 hardware is the original board for programming LEGO-based applications. It has an ARM9 processor that runs a Linux-based operating system. It has four output and four input ports. It is empowered with 16 MB of flash memory and 64 MB of RAM. It is also possible to increase the memory capacity up to 32 GB. As antenna hardware, it can communicate with Wi-Fi and Bluetooth dongles.

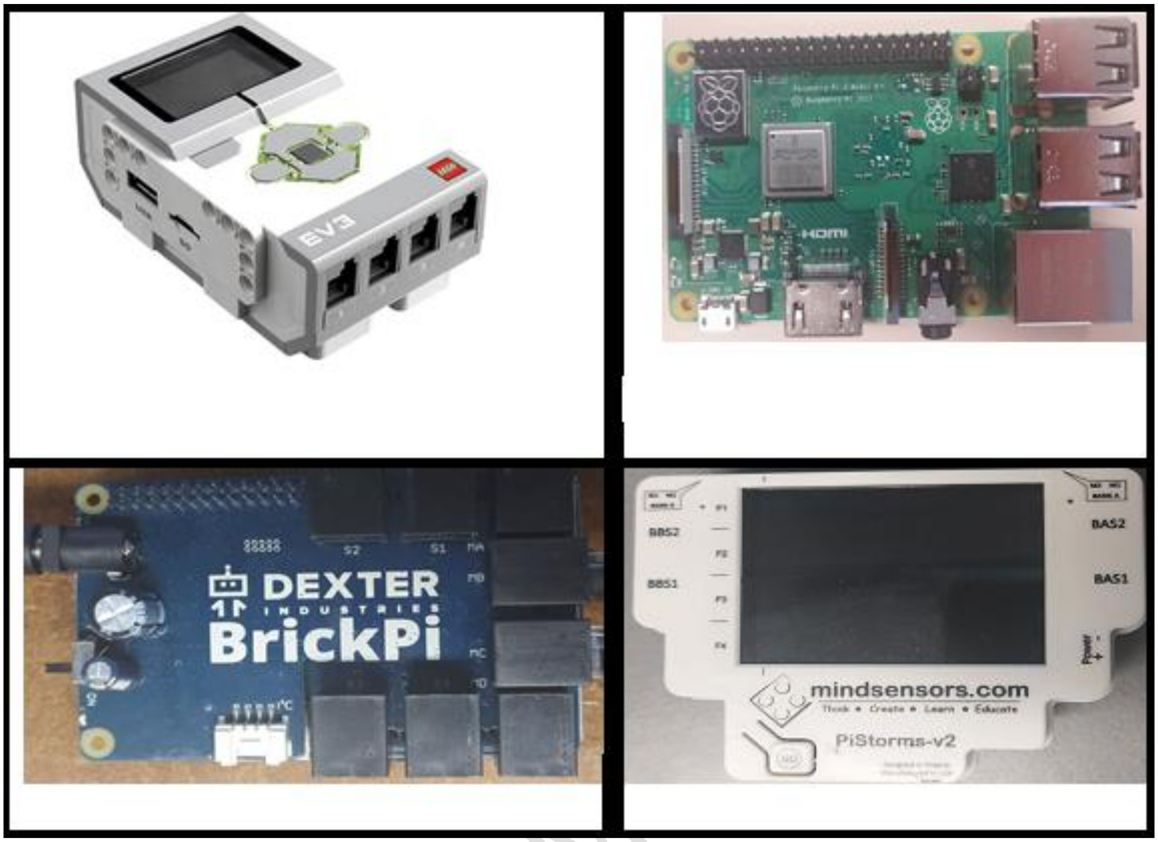


Figure. 9.2: LEGO Adapted Hardware.

**RaspberryPI 3:** As represented by the right top side of Figure 9.2, the Raspberry Pi 3 is a credit card size low powered computer board with Ethernet and Wi-Fi connection. It has an HDMI video output, an audio output and an SD card slot. The RaspberryPi is beneficial hardware for high-end tasks.

**BrickPI Hardware Interface:** As represented by the bottom left side of Figure 9.2, the BrickPi is a hardware interface for RasbperryPi that allows controlling LEGO sensors and actuators. It is attached to the top of the Raspberry Pi via hardware pins to work with LEGO technology. It has four input and four output ports connecting LEGO sensors and actuators. It can work with RaspberryPI's Wi-Fi and Bluetooth.

**PiStorms-v2 Hardware Interface:** As the right bottom side of Figure 9.2 shows, PiStorms-v2 is a LEGO-compatible interface for Raspberry Pi 3 (Yalcin et al., 2021). It enables to control of LEGO sensors and actuators when it is attached to a RaspberryPI 3 board. Similarly, it also has four input and four output ports for connecting LEGO sensors and actuators. It can work with RaspberryPI's Wi-Fi and Bluetooth.

**Agent Development Frameworks:** In this section, agent development frameworks such as Jason, JADE and SPADE agent platforms used during the implementation of the agent-based CPSs were discussed briefly.

**Jason BDI Agents:** Jason is a prolog-like logical programming language of Agentspeak and an extended interpreter version of the Java environment (Bordini and Hübner, 2005). Agentspeak language was established on the well-known Procedural Reasoning System (PRS) architecture which explicitly embodies the BDI model. In BDI, agents continuously observe their environment and react instantly to the changes in the environment.

**JADE Agents:** JADE is an agent-programming framework that facilitates the development of multi-agent systems. JADE is a distributed agent development framework with a flexible infrastructure that allows extensions based on Java. It has a run-time environment where JADE agents can be created and live within the given host and device. Developers can directly specialize these agents according to the requirements of their system needs.

**SPADE Agents:** SPADE is an agent development platform that allows the creation of multi-agents using Python language (Palanca et al., 2020). It is built using a new communication framework, namely Jabber, which provides new capabilities to the communication layer. A SPADE agent can run multiple tasks simultaneously. Each SPADE agent can obtain more than one task.

## 9.5 Architecture

The proposed architecture aims to create a CPS using integrated hardware and a single programming platform that covers both cyber and physical parts. This architecture can be used as a reference as different architectures have also been proposed in the literature (Karaduman et al., 2022). The architecture represented in Figure 9.3 is proposed considering the requirements of deploying agents onto an embedded system. There are six layers, and each of them is explained. The first three layers can be inspected under the physical side, and the last three can be grouped under the cyber side. At first, physical layers are mentioned.

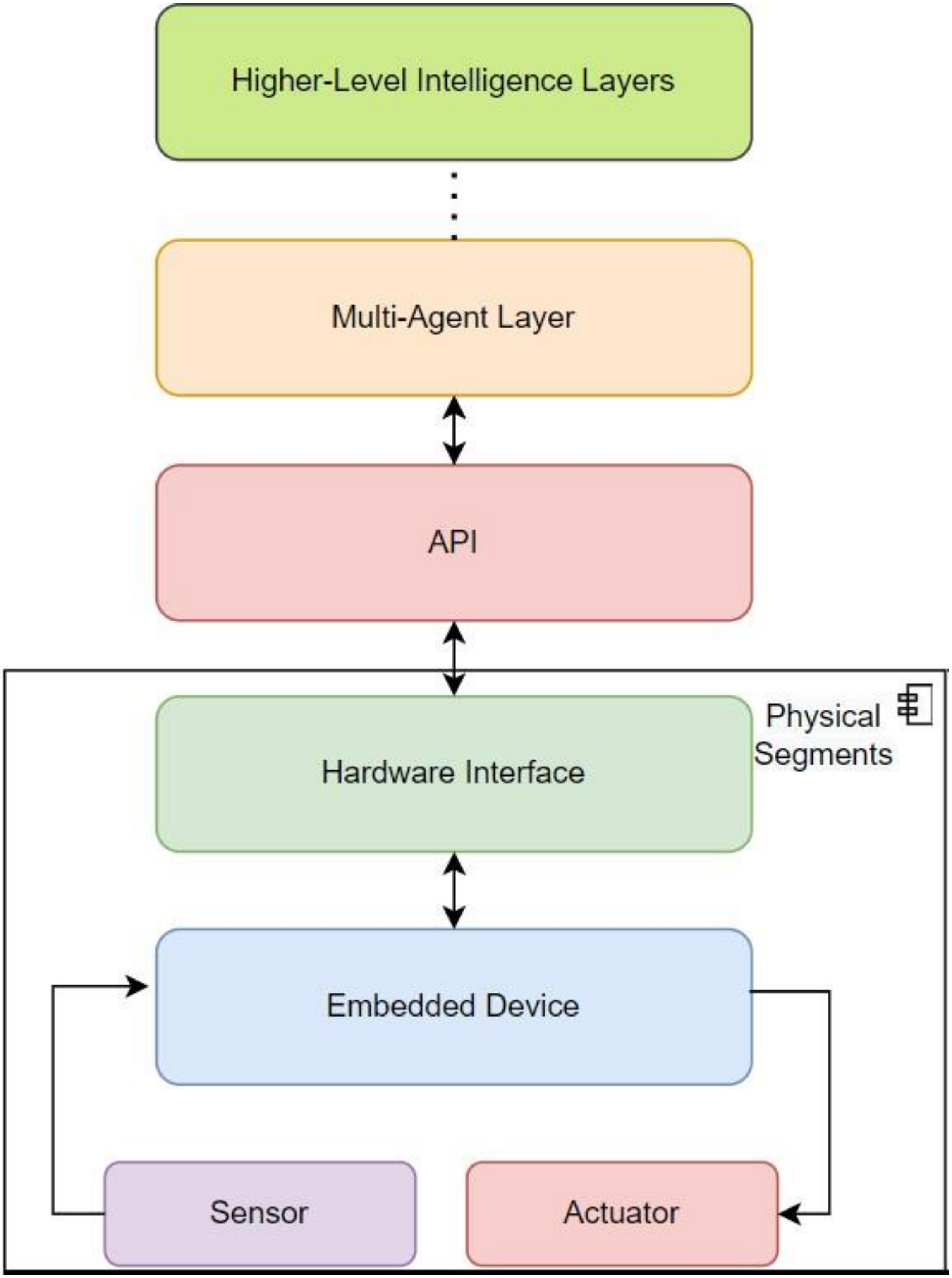


Figure. 9.3: Proposed Architecture for implementing CPS.

**Layer 0: Physical Segments** This layer includes all physical layers and components of a CPS. The segments can consist of many passive and physical type entities. They conform to physics laws. A set of passive components create a segment. Our study addressed this layer using LEGO technology and composable parts made of assembled plastic bricks.

**Layer 1: Sensor and Actuator** Layer The sensor and actuator layer describes the physical environment where the CPS was located. A CPS changes its environment using its physical capabilities. These changes create events via actuators around that environment; events created by the environment are perceived via sensors and actions.

**Layer 2: Embedded Device** layer contains a microcontroller development board where the sensors and actuators are connected and controlled. Sensors are used to gather data from the environment, while actuators are used to create motion and/or movement. However, the selected hardware should be capable of running the required software in the cyber layer. Furthermore, it should have network ports, especially wireless ones such as Wi-Fi and Bluetooth-Low Energy (BLE).

**Layer 3: Hardware Interface** is preferred to increase the capabilities of the embedded device. It can be an extension board, HAT, or expansion interface to adapt the embedded device to the system's requirements. It is an optional layer but essential in augmenting the hardware's capabilities. This hardware is not a computation device. They are meant to adapt the embedded hardware's peripheral interface for a particular hardware technology or domain. Cyber layers of the provided architecture are discussed in the following paragraphs.

**Layer 4: The API** layer describes any device-specific software. This API should provide device functions to set, configure and control the sensors and actuators. It should also abstract away the hardware level programming details such as device registers, bit shifting, bit-wise operations and instruction-based operations.

**Layer 5: Multi-Agent Layer** is a middleware where the agent framework runs. The MAS layer controls the device-specific functions. In this way, agents' perceptions are bound to the sensors and agents' actions are attached to the actuators of the embedded device. Agents can be reactive, cognitive or hybrid. Therefore, this layer contains behavioural definitions, plans, goals, and beliefs. Since an agent-based framework is independent of any embedded device, it should be merged with device API to be deployed into the embedded hardware.

**Layer 6: Higher-Level Intelligence Layers** represent where the higher-level intelligence is contained. This layer applies logic-based approaches, machine learning techniques and probabilistic approaches. Generally, these higher-level methodologies are computationally heavy and provide long-term results.

### 9.6. Development Workflow

In order to describe the process that has been conducted to construct all agent-based CPSs (which will be discussed later in the chapter), a workflow has been provided. Figure 9.4 illustrates three swim-lines: a cyber, a physical and a documentation column. The workflow begins with the documentation phase, where the UML/SysML diagrams are created. These diagrams are used to describe the system architecture at the model level. The system architecture can be represented by Block Diagrams (Challenger et al., 2011; Challenger et al., 2016). There should be at least one embedded device to create the cyber side of the system. The cyber side can be considered the abstract container for the software.

In this way, software entities such as software agents can deploy. Agents can be modelled using activity diagrams considering the model elements which were created a step before. Each activity can represent the behaviour of an agent. The physical form of a CPS (plant) is shaped using a sensor, actuator, and passive physical components. The passive physical components are merged with actuators and/or each other to create the planned motion, rotation, and movement. For example, a conveyor system is designed using a lot of combined belts to be formed. A motor then creates the necessary rotational movement to move the conveyor system. We chose LEGO to provide the physical construction of the agent-based CPS. We prioritized the structure of the physical system because the software that will run on the cyber side should be developed according to the capabilities of the physical system. For example, in a production line system, the software should be shaped considering the process phases and quantities of the physical sensors actuators. Specifically, how many motors should be configured and controlled by the agents and which type of sensors will be used should be decided before finalizing the code. After building the plant, the base code is developed. This base code is the software that runs the low-level API code fragments encapsulated into the preferred language's functions. However, it might not be possible to achieve seamless control over the actuators and sensors at first. The requirements such as the speeds of the motors, operation angle of motors and prepossessing of the sensors should be calibrated as another activity. At the same time, the base code is used for controlling physical components. Once the calibration of the components is completed, the code can be deployed onto embedded hardware. Each component then should be tested to ensure it can realize the desired process(es). The code fragments tested and calibrated can be merged to establish low-level control. Once the low-level control definition is completed, the agents can be defined according to their roles. Agent behaviour should be defined for the planned task. Then, agent communication should be decided on which type of messages should be sent or broadcast to which agents. When the design is OK, the agents can be integrated with low-level functions. If the design is not OK, the low-level control should be checked and modified. Once agents are integrated with low-level functions, an integration test should be done. If the test fails, workflow should be followed again, starting from the update of the plant.

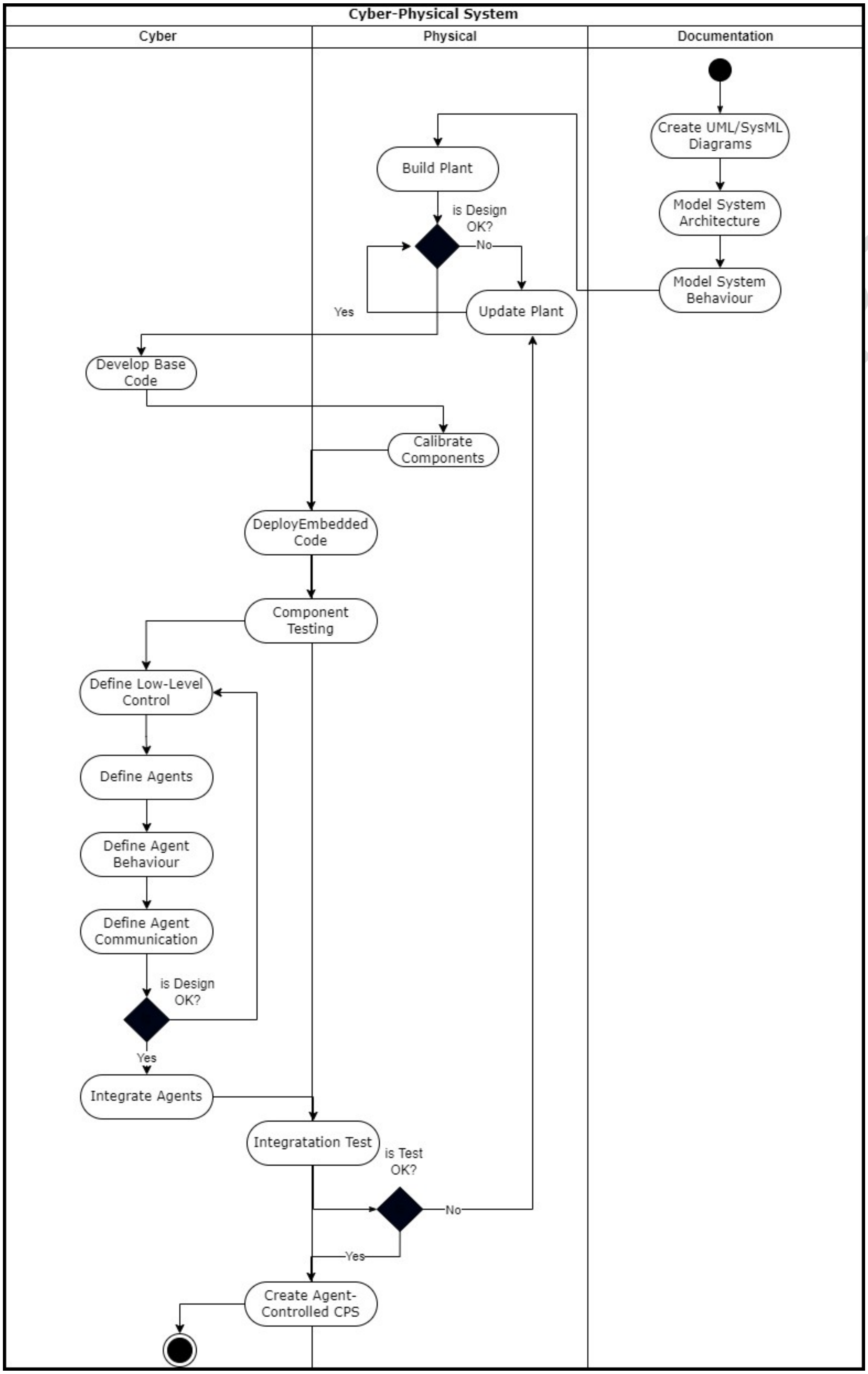

Cyber-Physical System
Cyber
Physical
Documentation
Create UML/SysML Diagrams
Model System Architecture
Model System Behaviour
Build Plant
is Design OK?
No
Yes
Update Plant
Develop Base Code
Calibrate Components
DeployEmbedded Code
Component Testing
Define Low-Level Control
Define Agents
Define Agent Behaviour
Define Agent Communication
is Design OK?
Yes
Integrate Agents
Integratation Test
is Test OK?
No
Yes
Create Agent-Controlled CPS

Figure. 9.4: Workflow for implementing the agent-based CPSs.

## 9.7 Concrete Implementation for the Agent-based CPSs

In this section, agent-based CPSs are described using the aforementioned technologies. The following platform comparison Table 1 is given to provide better insights.

Table 9.1: Hardware comparison table.

| **Features/ Board Name** | **EV3** | **BrickPI 3** | **PiStorms-v2** |
|---|---|---|---|
| Requires RaspberryPI | No | Yes | Yes |
| OS | Ev3Dev+ | Ev3Dev+ Debian | Ev3Dev+ |
| AOP | JADE | Jason, JADE | SPADE |
| OOP | Java | Java | Python |
| Use Case | ACC | LF, PL | Production Line |
| Button | 6 | 0 | 1 |
| Input Output | 4+4 | 4+4 | 4+4 |
| Support for EV3 Comp | Yes | Yes | Yes |
| Support for NXT Comp | Yes | Partial | Partial |
| Display | Yes | Addable | Yes |
| Has Interface | Yes | Yes | No |
| Battery Indicator | V,I | V | V |

Three examples of agent-based CPS studies are given in the following subsections. The convoy system has two robots: the line follower robot and an adaptive cruise control robot. In the convoy system, a robot system that tracks each other to mimic the production transportation of a manufacturing factory was proposed. The production line system imitates a product life cycle. The production line system takes a LEGO brick as an input, and that brick goes through multiple phases. In the convoy robot system, the ACC robot has positioned some distance behind the LF robot, and the LF robot is placed on a black line which is supposed to be tracked by the LF robot. Once the LF robot starts following the line, the adaptive cruise control detects the movement and starts to follow the LF robot while adapting its speed to keep a fixed distance. Two robots are implemented using different hardware to provide a heterogeneous and complex system. The LF robot is constructed using a RaspberryPi, the ACC is equipped with EV3 Brick, and a production line system was implemented using both PiStorms-V2 and BrickPI boards. In the following subsections, the implementation details, which conform to the workflow represented by Figure 9.4, were given based on the production line system. We preferred to select the production line system for brevity and size constraints of this chapter, but other examples also conform to the same workflow.

**Agent-based CPS Example 1:** The Production Line system represents a stationary, multi-stage, complex system (Yalcin et al., 2021). Each agent has its goals to be achieved. Once a goal is achieved, an inform message is sent to the target agent. All agents behave to mimic a product life cycle. It can be inspected under cyber-physical production systems (CPPS). CPPS requires dynamic control to enhance product quality with artificial intelligence, machine learning methods, and agent-based techniques. This system was implemented using six software agents, using one-shot, cyclic and FSM behaviour types. Eight actuators and three sensors were used (Karaduman et al., 2021). A demonstration video can be found on https://youtu.be/H1hbTqo0BBY. For brevity, the production line's push segment steps are addressed based on the workflow and the proposed architecture, but Figure 9.5 describes the final structure. Documentation activities mostly describe which components should be used according to the proposed architecture, and cyber and physical activities describe the conformance relations to the introduced layers.

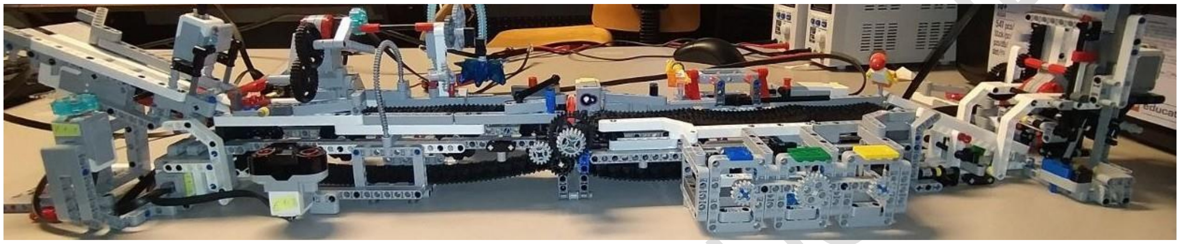


Figure. 9.5: Production Line System.

**Documentation Activity:** To model a CPS, suitable UML/SysML diagrams should be selected. For this case, the Block Definition Diagram was chosen for modelling the architectural hierarchy of the system, and the Activity Diagram was preferred for modelling the system's behaviour. The agents' communications were modelled using the Sequence Diagram. As the left side of Figure 9.6 represents a block definition diagram to model the second layer's architecture of the production line system. It shows the composable hierarchy from top-to-bottom.

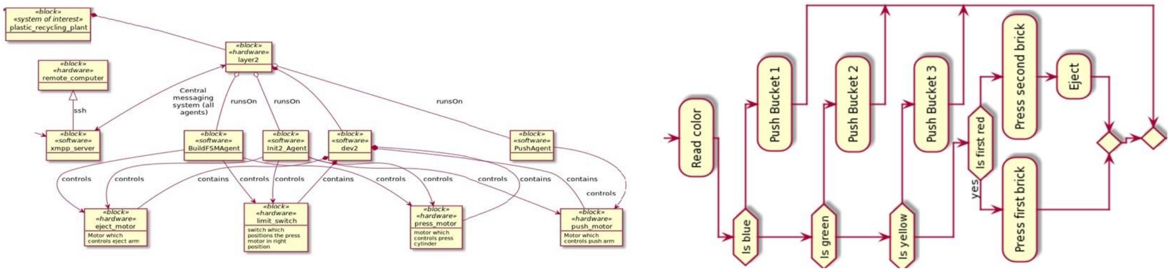

«block» «system of interest» plastic_recycling_plant
«block» «hardware» layer2
«block» «hardware» remote_computer
ssh
Central messaging system (all agents)
runsOn
runsOn
runsOn
«block» «software» xmpp_server
«block» «software» BuildFSMAgent
«block» «software» Init2_Agent
«block» «software» dev2
«block» «software» PushAgent
controls
controls
contains
controls
controls
contains
controls
controls
controls
contains
contains
controls
«block» «hardware» eject_motor
Motor which controls eject arm
«block» «hardware» limit_switch
switch which positions the press motor in right position
«block» «hardware» press_motor
motor which controls press cylinder
«block» «hardware» push_motor
Motor which controls push arm
Read color
Is blue
Push Bucket 1
Is green
Push Bucket 2
Is yellow
Push Bucket 3
Is first red
yes
Press second brick
Press first brick
Eject

Figure. 9.6: Excerpts of the UML/SysML diagrams.

The right side of Figure 9.6 describes an excerpt from the Activity Diagram. It shows the activity that the Push agent follows. According to the colour of a brick, it pushes that brick to the corresponding bucket. If the colour is red, the Build agent presses the red bricks. The agents update each other for contextual changes using communication. According to the content of the message, an agent can actuate a motor and/or start sensor sampling.

**Physical Activity:** When the modelling steps are finished, the plant can be built based on the models provided for building the plant. At first, the plant should be built using the physical segments while combining these physical segments, which consist of composable components. At first, the motors and sensors should be selected according to the system's models and layer 1 of the proposed architecture. Then, chosen sensors and actuators should be merged with Physical Segments (level 0), which consist of passive LEGO components. The proposed architecture's layer 0 includes all the physical layers. Therefore, it should be built first. For example, the left top side of Figure 9.7 illustrates the construction process of the Build segment of the production line system. The right top side of Figure 9.7 depicts the Shred section of the production line. It is another segment that constructs the plant. The bottom left of Figure 9.7 shows that these segments are merged to build the whole plant. If necessary, some updates, such as adding extra components or reducing the length of a part, can be applied.

**Cyber Activity:** Once the plant's construction is finished, a base code planned to control the physical part can be implemented as a cyber activity. We used a RasperryPI 3 as an embedded device and BrickPI 3 as a hardware interface. The base code creates a substance for reading the data from the sensors and controlling a motor, including necessary configurations to use the API functions, which is the next required layer. However, they are the code fragments that influence these components and cannot be used to create process chains until correct calibration parameters are found. An extra step should be followed to find the exact operation parameters.

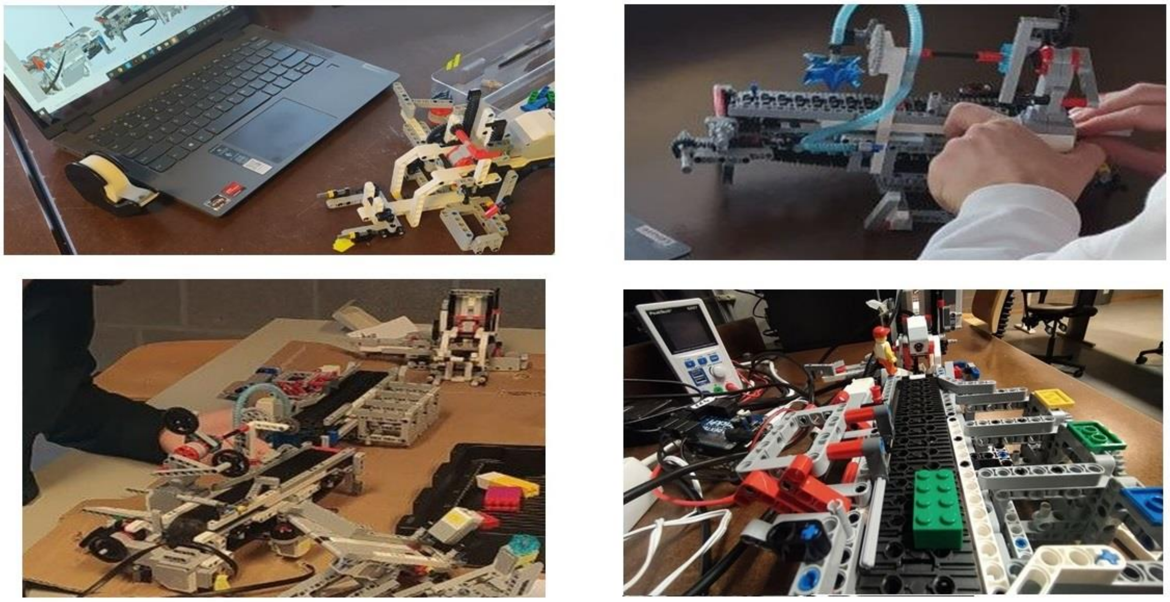


Figure. 9.7: Construction steps of a CPS using LEGO Technology.

Moreover, API functions' parameters can also be inputs for setting speed, positioning degree, arranging sensor sampling rate etc. However, it may not be possible to find ideal calibration parameters simultaneously. Therefore, it requires an iterative trial and error approach and observance of the physical activity to develop a focus on the movement range. A physical system that creates motion, rotation or movement should have limits for the freedom of displacement. In other words, the motion created by a LEGO motor should be realized in the range of any points between the physical limitations of that component. For example, the push mechanism that pushes the buckets into the correct buckets requires a valid operation range and parameters. Therefore, the physical operation range can be found by applying the trial-and-error method. Specifically, the parameter values can be increased or decreased according to the achieved task. The right bottom side of Figure 9.7 illustrates the push mechanism and green brick. The figure shows that the green brick should stop in front of the middle bucket but could not be stopped at the correct location. Therefore, the rotation duration of the conveyor motor should be increased to stop the brick at a suitable spot.

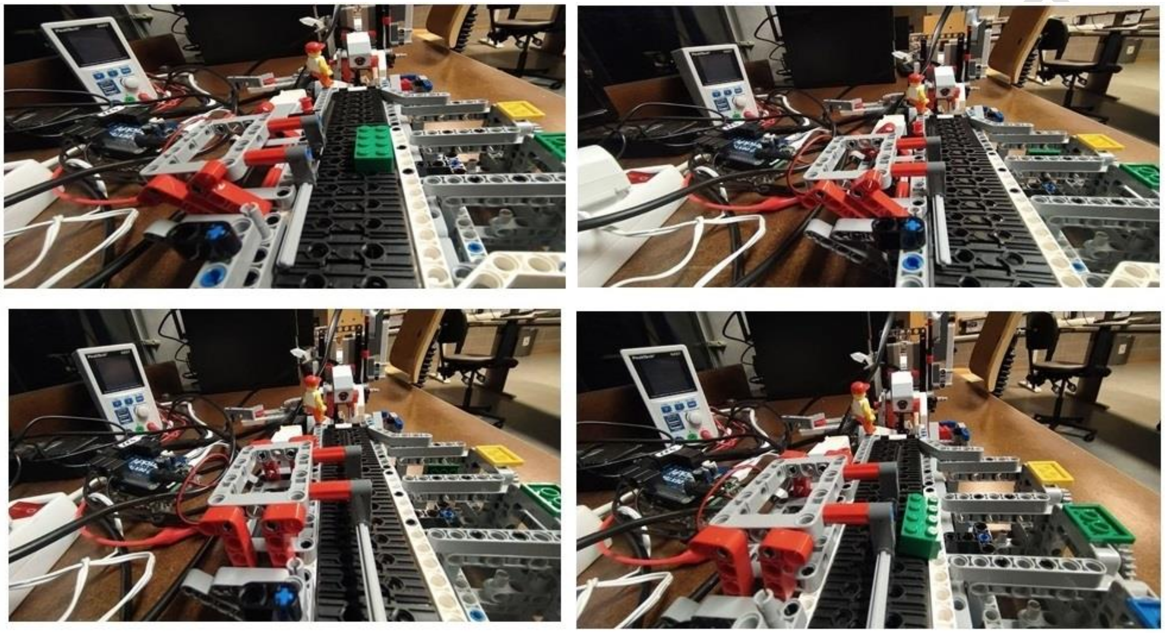


Figure. 9.8: Calibration steps of the Production Line.

The left top side of Figure 9.8 shows that the green brick was able to be stopped at the correct location. This action was achieved after multiple trial and error efforts, and the resulted function parameters were selected as the operation(calibrated) parameters. The right top side of Figure 9.8 depicts the initial position of the push mechanism. It should return to its initial position after each pushing action. Therefore, the parameter for the initial position is 0 and is the reset parameter. At first, a rotation degree to the middle point was found to calibrate the push mechanism. Then rotation torque was calibrated. Because the displacement time is too much, then it is possible to throw the brick away instead of pushing it into the bucket. The left bottom side of Figure 9.8 illustrates that the push mechanism only moved to the middle of the conveyor belt. The solution can be increasing the torque and increasing the operation length. In this way, the pushing mechanism can touch the brick and pushes it. The right bottom side of Figure 9.8 illustrates that the pushing mechanism moves a little bit further from the conveyor belt's middle point but fails to put the green brick into the bucket because of low torque. As a result, we decided to move the pushing mechanism until its mechanical limits with an intermediate-level torque. Eventually, we set the push mechanism to have the correct configuration and succeeded in pushing the green brick into the bucket.

**Cyber-Physical Activity:** When all the components' calibration is completed, the embedded code which uses the operation parameters can be deployed onto the embedded hardware. Once the code is deployed, each segment's components should be tested to ensure they can work in a combinational manner. This way, components' functionalities and synchronization between them are also tested. In other words, the interplay and influential effect between cyber and physical entities are observed.

**Cyber Activity:** When the test phase is completed, cascade low-level operations can be merged. For example, at first, a motor can be assigned to a port, namely portB. It can then be actuated for 200ms using a speed parameter of 200 units. After this, it is stopped for 100ms. Another function, namely "moderate operation", can use different parameters, stop/start operations and platform-specific API functions. In listing 1.1, calibrated functions were given. These functions can be called in a sequential or combinational manner. In this way, the necessary movements of the components can be achieved. In addition, sensor reading timings can also be arranged. The sensor reading can be activated once a specific event is realized or based on a period. For example, when the button is pressed, the colour sensor can be activated to check whether a brick has arrived or not. As agent-oriented programming proposes higher-level abstraction to develop software, software agents were defined according to the individually quite complex sections. As the proposed architecture's layer 5 presents, the agents can be determined according to the sections to be controlled. The processes and events that should be handled on the cyber side can be defined using the agent's behaviours. The agents' communication should be specified on the cyber side. The sequence diagram can be used to model the communication between the agents. This way, when an event occurs or ends, the agents can inform each other to trigger other events. When this step is finished, the design should be checked. If the design is not OK, it should be re-planned starting from the low-level control. Once the design is agreeable, the design can be implemented, and agents can be integrated with the low-level system functions via their behaviours and establish communication among them.

**Cyber-Physical Activity:** The integration test should be applied when the integration is finished. Physical events should be created to realize the processes. Agents are expected to control the system by realizing the required actions and sensing the environment. If the test fails, the plant should be updated, and the previous steps should be re-checked. If the test succeeds, an agent-controlled cyber-physical system is created. As a result, an autonomous agent-controlled CPS is established based on the workflow and the proposed architecture. Intelligence methods can be applied to this exemplar system to merge the higher layers.

**Agent-based CPS Example 2: Line Follower Robot:** The Line follower is a robot that follows the black lines to track a pre-defined line. It has two agents, Motor and LineFollower (Schoofs et al., 2021). It has a power button to be powered on or off. When it is initialized to be functional, it starts sensor reading and sends the data to the Motor Agent. According to the incoming data, the motor agent decides to turn left, right, stop or slow down. A decision is taken when the line follower robot encounters a turn. To make the turn, it slows down, turns and speeds up. While it realizes these actions, it sends the same action parameters to the Adaptive Cruise Control agent. Line Follower agent has three cyclic behaviours for establishing communication, controlling the motors and data sampling. We also benefited from SimpleBehaviour and TickerBehaviour. Four sensors and two motors were used to implement this system. SimpleBehaviour and TickerBehaviour were also utilized.

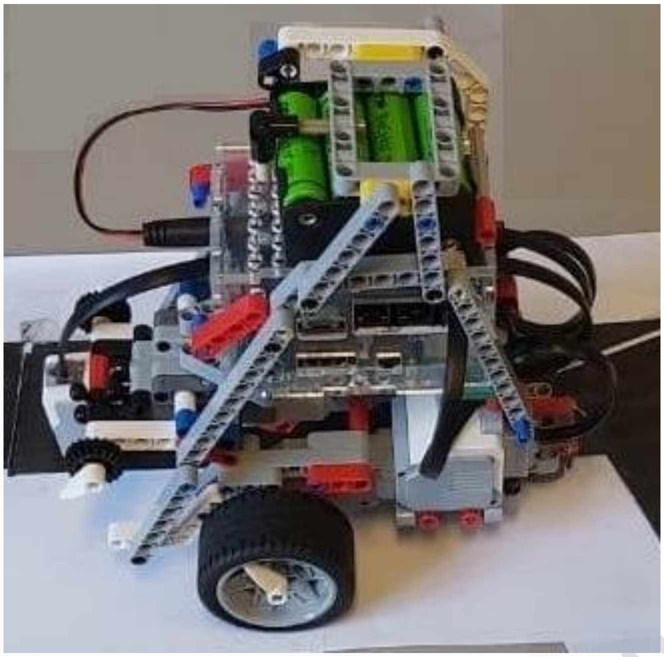


Figure. 9.9: Line Follower Robot.

**Agent-based CPS Example 3: Adaptive Cruise Control Robot:** The Adaptive Cruise Control Robot (ACC) robot has an ultrasonic sensor to detect the distance between LF robot's (Schoofs et al., 2021). When the LF robot is taking a turn, the ACC robot receives messages from the LF robot to adapt its motion to have the same turn. The ACC robot estimates the sharpness of the turn and tries to realize turning actions synchronously. It starts its operation when the power button is pressed. It then starts to receive messages and sample sensor data used for distance measurement, relative speed computation and arranging current speed. When a message is received, the ACC robots try to take a turn. This system was implemented using two motors and one sensor. Moreover, it benefited from two CyclicBehaviour, one SimpleBehaviour and one OneShot Behaviour.

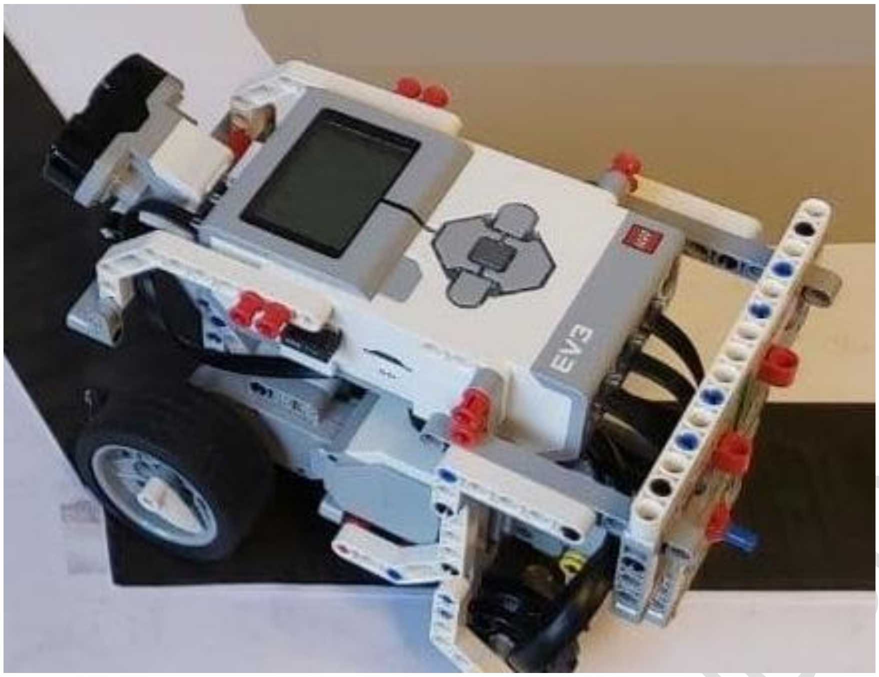


Figure. 9.10: Adaptive Cruise Control Robot.

## 9.8 Software Excerpts

In this section, some code excerpts were given. During the implementation of this agent-based CPSs, singleton design pattern and static class definition approaches were selected. The singleton design pattern has been applied to encapsulate device-level functions into agents' behaviours. Because multiple object creations have been encountered within two different embedded hardware and agent programming platforms, the resource access problem had occurred considering that software agents are concurrent entities. Therefore, the problem was tackled by limiting object instantiation. Alternatively, static function definitions could also be used. Thus, this concludes that singleton design pattern or static function definitions may be required while integrating device functions with software agents. As Jason uses a prolog-like language and BDI structure, it does not require such an approach.

Table 9:2 Code Excerpt from the examples.

```
+!checkButtonStatus   :  dropButtonStatus ( false )  <− buttonPressed ;  ! checkButtonStatus .
+!checkButtonStatus   :  dropButtonStatus (true)  <−!checkProductStatus .
public  class  Button {
    public  static  boolean  isPressed ()  {
    EV3TouchSensor touchSensor = new EV3TouchSensor(SensorPort.S2);
     boolean touch  = InitComp.touchSensor. isPressed () ;
     return  touch;}}
  psm = PiStorms() __instance  = None
    @staticmethod def  getInstance () :
```

```
10          if  dev1.__instance  == None: dev1() return  dev1. __instance
11 def  __init__( self ) : if  dev1. __instance  != None: raise  Exception(" Singleton  class ") else :
dev1. instance  =  self
```

Table 9.1's lines 1 and 2 describe a plan excerpt, checkButtonStatus, from the Jason implementation. If button is pressed by the user, the dropButtonStatus(false) becomes dropButtonStatus(true). The first line works until the condition of the button is changed, and in each agent cycle, the state of the button is checked using the buttonPressed action. If the status becomes true, then checkProductStatus plan is triggered. The low-level implementation of the button control code is represented between lines 3 and 7. Using a CyclicBehaviour of the JADE, the state of the button is checked in each cycle. The singleton pattern used in the SPADE implementation is shown in lines 8 and 10. This pattern was used to constrain the object creation to only one because accessing two different objects caused inconsistency in controlling the low-level API.

## 9.9 Discussion & Technical Notes

The adaptation of MAS to CPS is an open research domain. This book chapter presents concrete agent-based CPSs and their corresponding requirements to show how Agent-based frameworks can be deployed into various embedded hardware. As recently, embedded system programming interests have shifted to more abstract OOP-based languages such as microPython and C++. At the same time, it is still benefited from high-level languages such as C and Basic. We are motivated to show how software agents can be used to program CPSs as they have a higher abstraction compared to OOP and other high-level languages.

Moreover, we are motivated to provide design choices, an integrated architecture, and concrete agent-based CPSs to put sheds for rapid prototyping of both the cyber and physical sides of CPS. As CPS can be enhanced with software agents, model-driven approaches can also be an integral paradigm to reduce complexity (Challenger et al., 2021; Challenger et al., 2020). In this regard, a casual language based on ABM and MAS should be implemented to represent the structure and behaviour of the target dynamic system. These elements should cover both design-time and run-time elements. It is necessary to develop new solutions and adapt existing standards such as IoT, machine learning and fuzzy logic (Karaduman et al., 2021).

Moreover, we would like to share some technical details that may be beneficial for practitioners and researchers. Switches can be used to limit the movement of dynamic parts. Once the moving part or platform touches the limit switch (i.e., utilizing a LEGO button), the system can detect the state change and stop the movement. While implementing the agent-based CPSs, we have seen that preferring one agent for each section or robot is more suitable for development time and code complexity. This way, the separation of concerns approach can be followed to avoid accidental complexity and mutual exclusion problem. A suitable power source should be selected according to the system type, such as mobile or stationary. For a stationary system, tunable power supplies can be used along with the parameters of 9.8 Volts and 3 Amps. These are the common parameters to feed all the hardware types because of LEGO EV3 motors and sensors' power requirements. Two Li-Po batteries (18650) can be used for a mobile system when the PiStorms-V2 is preferred. However, a stationary system can also use the same voltage and current levels (i.e., 9.8 Volts and 3 Amps). In our agent-based CPSs, we used 8xAA 2600mAh rechargeable batteries when the BrickPI had selected as programmable hardware. They provide enough energy for approximately 2 hours. In case of low battery, at first, motors stop working, but sensors may remain operational. This may confuse the developer, which can take time to find the source of the cause. Therefore, batteries should be checked first when the system is mobile. When the motors run for a long time at high speed, they may heat up, so cold spray or cold surface gels should be applied. The friction between moving parts can be reduced using oil. However, because of LEGO's material, too much machine oil sticks and does not dry for a long time. Therefore, a lightweight type of oil should be preferred. While working with the PiStorms-V2, the display did not work correctly, but it did not affect our implementation.

Moreover, sometimes the power button was also disabled. We disabled the power source for a few seconds, and then it got operational. On the BrickPI side, we have seen that switching off BrickPI's onboard power switch caused batteries to get short-circuited. It created some smoke and damaged the batteries. Therefore, we stopped using the power switch and plugged out the battery jack when needed. The LEGO sensors may require additional configurations. For example, the colour sensor cannot recognize all tones of colour. Therefore, specific ones should be found. If a brick or part moves fast, the colour sensors' sampling time might not be enough to recognize it, or the wrong colour can result.

Furthermore, the ultrasonic sensor's range might be too far. Therefore, the operation distance can be limited from the software part. Moreover, it can get the maximum integer value when the distance is infinitive. Thus, these arrangements should be detected in the Develop Base Code step and calibrated in the Calibrate Components phase using the trial-and-error method. Then calibrated parameters should be used to perform necessary physical actions. A mechanical system can be modelled according to parameters such as mass, length, the material of the components, and time. Some assumptions must be made to compare large and complex systems with their scaled-down representations. In addition, LEGO technology provides a physical abstraction by scaling down the systems, but it suffers from some parameters such as weight, friction, and durability. However, it creates an excellent form to mimic the processes, functionalities and behaviour considering steady-state conditions of an industrial system which can also be the asymptotic phase of most industrial systems. As mentioned, the trial-and-error method helps achieve the timings of industrial systems. Using LEGO technology, it may also be possible to implement actual operation timings or constant-based relative time periods on the miniaturized system. Then, the developed software can be integrated into the existing system by tuning the timing parameters.

## References

Arslan, S., Challenger, M., Dagdeviren, O.: Wireless sensor network based fire detection system for libraries. In: 2017 International Conference on Computer Science and Engineering (UBMK). pp. 271–276. IEEE (2017)

Asici, T.Z., Karaduman, B., Eslampanah, R., Challenger, M., Denil, J., Vangheluwe, H.: Applying model driven engineering techniques to the development of contiki-based iot systems. In: 2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT). pp. 25–32. IEEE (2019)

Baheti, R., Gill, H.: Cyber-physical systems. The impact of control technology 12(1), 161–166 (2011)

Bordini, R.H., Hübner, J.F.: Bdi agent programming in agentspeak using jason. In: International workshop on computational logic in multi-agent systems. pp. 143–164. Springer (2005)

Carreira, P., Amaral, V., Vangheluwe, H.: Multi-paradigm modelling for cyber-physical systems: Foundations. In: Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems, pp. 1–14. Springer (2020)

Challenger, M., Erata, F., Onat, M., Gezgen, H., Kardas, G.: A model-driven engineering technique for developing composite content applications. In: 5th Symposium on Languages, Applications and Technologies (SLATE'16). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)

Challenger, M., Getir, S., Demirkol, S., Kardas, G.: A domain specific metamodel for semantic web enabled multi-agent systems. In: International Conference on Advanced Information Systems Engineering. pp. 177–186. Springer, Berlin, Heidelberg (2011)

Challenger, M., Tezel, B.T., Amaral, V., Goulao, M., Kardas, G.: Agent-based cyber-physical system development with sea_ml++. In: Multi-Paradigm Modelling Approaches for Cyber-Physical Systems, pp. 195–219. Elsevier (2021)

Challenger, M., Vangheluwe, H.: Towards employing abm and mas integrated with mbse for the lifecycle of scpsos. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. pp. 1–7 (2020)

Karaduman, B., David, I., Challenger, M.: Modeling the engineering process of an agent-based production system: An exemplar study. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 296–305. IEEE (2021)

Karaduman, B., Oakes, B.J., Eslampanah, R., Denil, J., Vangheluwe, H., Challenger, M.: An architecture and reference implementation for wsn-based iot systems. In: Emerging Trends in IoT and Integration with Data Science, Cloud Computing, and Big Data Analytics, pp. 80–103. IGI Global (2022)

Karaduman, B., Tezel, B.T., Challenger, M.: Towards applying fuzzy systems in intelligent agent-based cps: A case study. In: 2021 6th International Conference on Computer Science and Engineering (UBMK). pp. 735–740. IEEE (2021)

Karnouskos, S., Leitao, P., Ribeiro, L., Colombo, A.W.: Industrial agents as a key enabler for realizing industrial cyber-physical systems: Multi-agent systems entering industry 4.0. IEEE Industrial Electronics Magazine 14(3), 18–32 (2020)

Leitao, P., Karnouskos, S., Ribeiro, L., Lee, J., Strasser, T., Colombo, A.W.: Smart agents in industrial cyber– physical systems. Proceedings of the IEEE 104(5), 1086–1101 (2016)

Palanca, J., Terrasa, A., Julian, V., Carrascosa, C.: Spade 3: Supporting the new generation of multi-agent systems. IEEE Access 8, 182537–182549 (2020)

Sakurada, L., Barbosa, J., Leitão, P., Alves, G., Borges, A.P., Botelho, P.: Development of agent-based cps for smart parking systems. In: IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society. vol. 1, pp. 2964–2969. IEEE (2019)

Schoofs, E., Kisaakye, J., Karaduman, B., Challenger, M.: Software agent-based multi-robot development: A case study. In: 2021 10th Mediterranean Conference on Embedded Computing (MECO). pp. 1–8. IEEE (2021)

Türk, E., Challenger, M.: An android-based iot system for vehicle monitoring and diagnostic. In:

26th Signal Processing and Communications Applications Conference (SIU).pp.1–4. IEEE (2018)

Yalcin, M.M., Karaduman, B., Kardas, G., Challenger, M.: An agent-based cyber-physical production system using lego technology. In: 2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS). pp. 521–531. IEEE (2021)