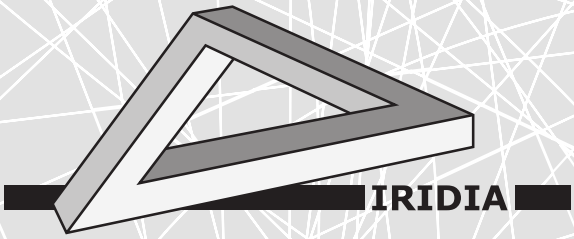


Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Description of the Electronics, Mechanical Design, and Software for an Autonomous Construction System using Stigmergic Blocks

M. Allwright, N. Bhalla, and M. Dorigo

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2017-007

March 2017
Last revision: March 2017

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2017-007

Revision history:

TR/IRIDIA/2017-007.001 March 2017
TR/IRIDIA/2017-007.002 March 2017

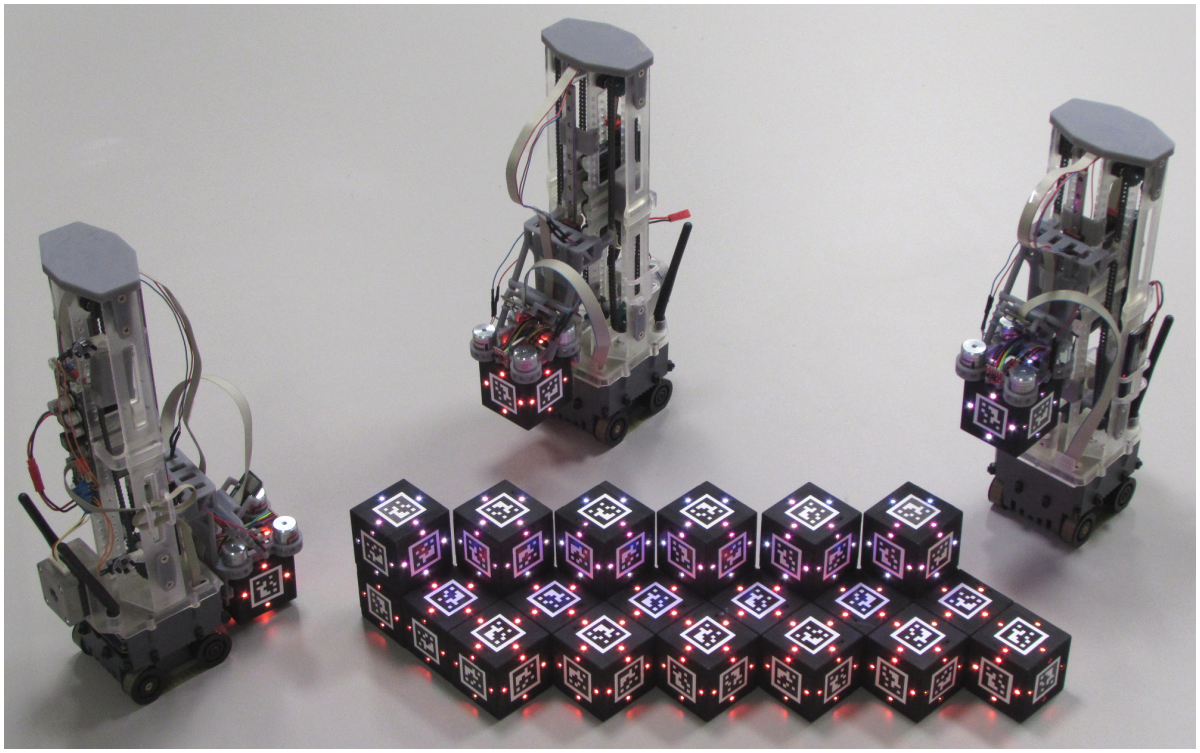


Figure 1.1: Three autonomous robots build a wall from stigmergic blocks

1 Abstract

In this document, we summarize the hardware implementation of our multi-robot construction system, which consists of stigmergic blocks and autonomous robots (Figure 1.1). The following sections detail the electronics, software, and mechanical design of these two components. This document is part of the supplementary material available for [1]. The accompanying videos for this supplementary material can be found online[1].

[1]Website: `http://iridia.ulb.ac.be/supp/IridiaSupp2017-004/index.html`

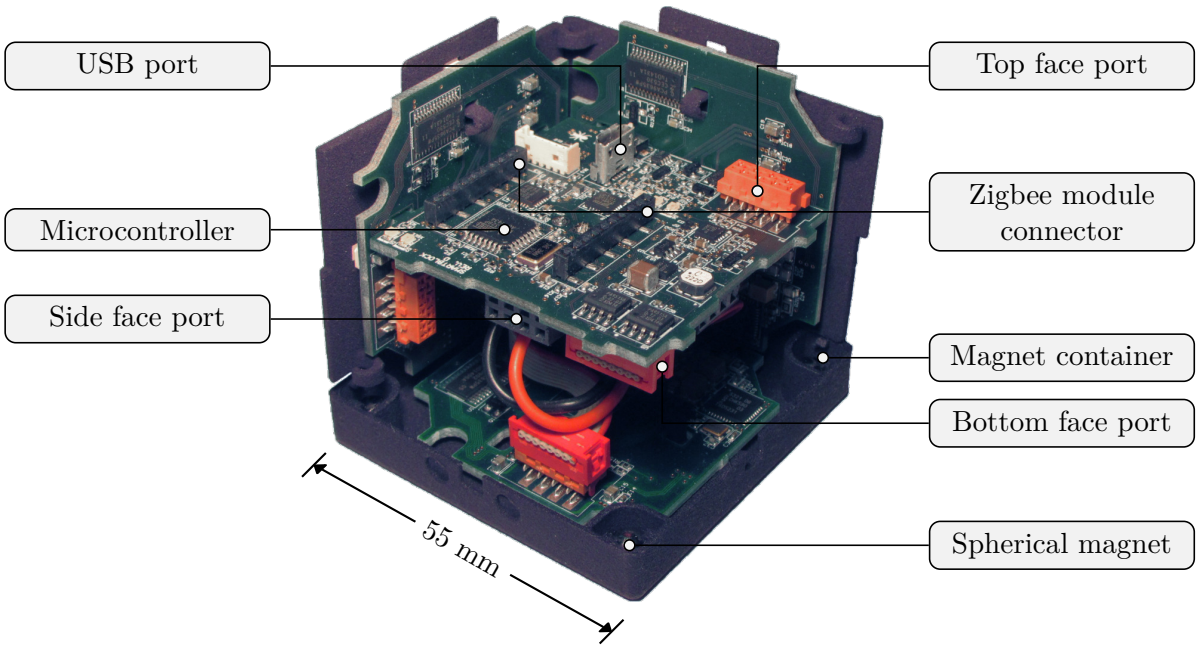


Figure 2.1: Internal view of a stigmergic block

2 Stigmergic Blocks

We have designed the stigmergic blocks to simplify the actuation and sensing requirements of the autonomous robots so that we can focus our work on developing decentralized control strategies for multi-robot construction.

In brief, a stigmergic block is an advanced cubic building material capable of computation, data storage, and communication (Figure 2.1). There are currently two types of communication used in the proposed construction scenarios. The first type uses a near field communication (NFC) interface to enable robot-to-block communication. The second type utilizes LEDs on the faces of a block to enable block-to-robot communication, which is facilitated by robot vision. The electronics of a block is implemented using a central circuit board and six face circuit boards. The exterior of a block has a side length of 55 millimeters and is printed using selective laser sintering.

The simplification of the actuation and sensing requirements is partially achieved by attaching the localizable tags presented in [2] to the stigmergic blocks, which enable the robots to accurately locate a block in an environment. Furthermore, we have added a freely-rotating, spherical magnet into each corner of a block to enable self-alignment and to reduce cumulative misalignment during construction. These spherical magnets also increase the structural integrity of a structure.

2.1 Electronics

At the core of a stigmergic block is a microcontroller[2], which runs a block's software. We have mounted the microcontroller on the central circuit board to manage power and the routing of data between various interfaces.

The microcontroller is programmed with the Optiboot bootloader[3], which enables reprogramming using the microcontroller's serial port. We have connected the serial port of the microcontroller to a USB-to-serial converter IC with USB battery charger detection[4]. This configuration allows for recharging, reprogramming, and debugging a stigmergic block over a single USB connection.

The USB-to-serial converter is configured over USB to provide control signals for a power management IC[5]. These signals inform the power management IC how much current may safely be drawn from the USB connection. The power management IC allocates this power to the system and to the attached lithium-ion battery for recharging. The system power is connected to two switching regulators, which can be switched on/off using the push button located near the USB port. The first regulator provides 3.3V for the digital electronics, while the second regulator provides 5V for the LEDs on the face circuit boards.

We provide a standard socket for a wireless module[6] on the central circuit board. The purpose of this wireless module is to enable remote debugging and monitoring of a stigmergic block. This wireless module connects to the microcontroller via a serial port. As the microcontroller only has one serial port, which is used for reprogramming via USB, a second serial port is emulated by using the 16-bit timer with the AltSoftwareSerial[7] library.

The central circuit board provides six connectors for each of the face circuit boards. These connectors provide each face circuit board with power, an interrupt line, and an I^2C bus. As the face circuit boards are identical, the I^2C bus is segmented so that there are no address conflicts.

Each face circuit board contains a near field communication (NFC) transceiver and an LED driver. The LED driver is used to set the brightness of the red, blue, and green channels of four multi-color LEDs on a face circuit board. An autonomous robot can sense the color of these LEDs from a distance while inspecting the containing structure. The NFC transceiver allows messages to be sent and received wirelessly to nearby robots or blocks.

[2] ATMega328P: `http://www.atmel.com/devices/atmega328p.aspx`

[3] Optiboot: `https://github.com/optiboot/optiboot`

[4] FT231X: `http://www.ftdichip.com/products/ics/ft231x.html`

[5] BQ24075: `http://www.ti.com/product/bq24075`

[6] Xbee Wireless Modules: `https://www.digi.com/lp/xbee`

[7] AltSoftwareSerial: `https://github.com/paulstoffregen/altsoftserial`

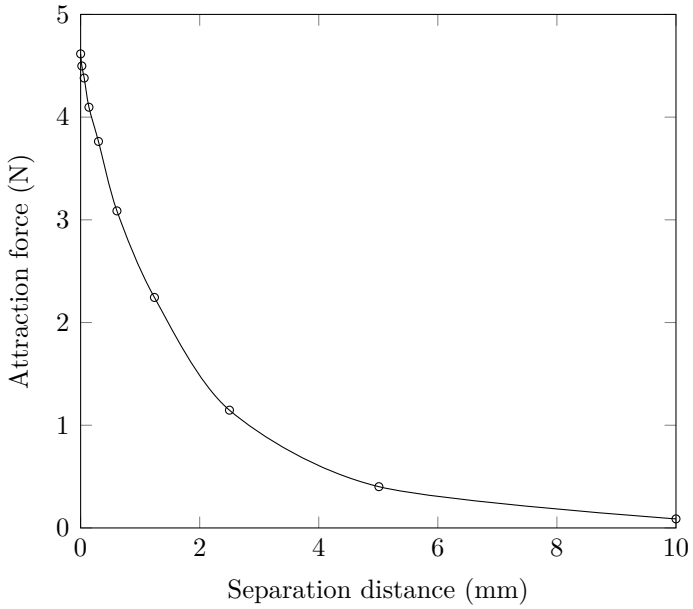


Figure 2.2: Plot of the attraction force between two spherical magnets

2.2 Mechanical Design

A stigmergic block is assembled from circuit boards, spherical magnets, and covers that we have printed using selective laser sintering. In this design, we use three types of covers: a side cover, a top cover, and a bottom cover. A stigmergic block is cubic in shape and has a side length of 55 millimeters.

Eight spherical neodymium magnets[8] are located in the corners of a block and enable a self-alignment characteristic, which also increases the structural integrity of a structure. These magnets are six millimeters in diameter and weigh 0.9 grams each. Figure 2.2 shows the attraction force between two of these magnets with respect to the separation distance.

Figure 2.3 shows the mechanical design of the different covers. We have designed the side covers to be used in an alternating up and down configuration. The top and bottom covers have their side cover slots and receptacles at different orientations to accommodate and align with the up and down configuration of the side covers. The side covers contain printed springs, which have been orientated so that the adjacent side covers are held in place using tension. This configuration provides the stigmergic block with structural integrity while allowing the top and bottom covers to be easily removed.

The top and bottom covers both contain four small insets for the spherical magnets. These magnets are held in place using small tabs on the sides of each inset, which allow a magnet to be inserted into position while remaining free to rotate. In addition, the top cover contains a small hole located above the power and reset switch on the central circuit board. This hole provides access to the switch using a small screwdriver.

[8]Datasheet for the magnets: `https://www.supermagnete.de/eng/data_sheet_K-06-C.pdf`

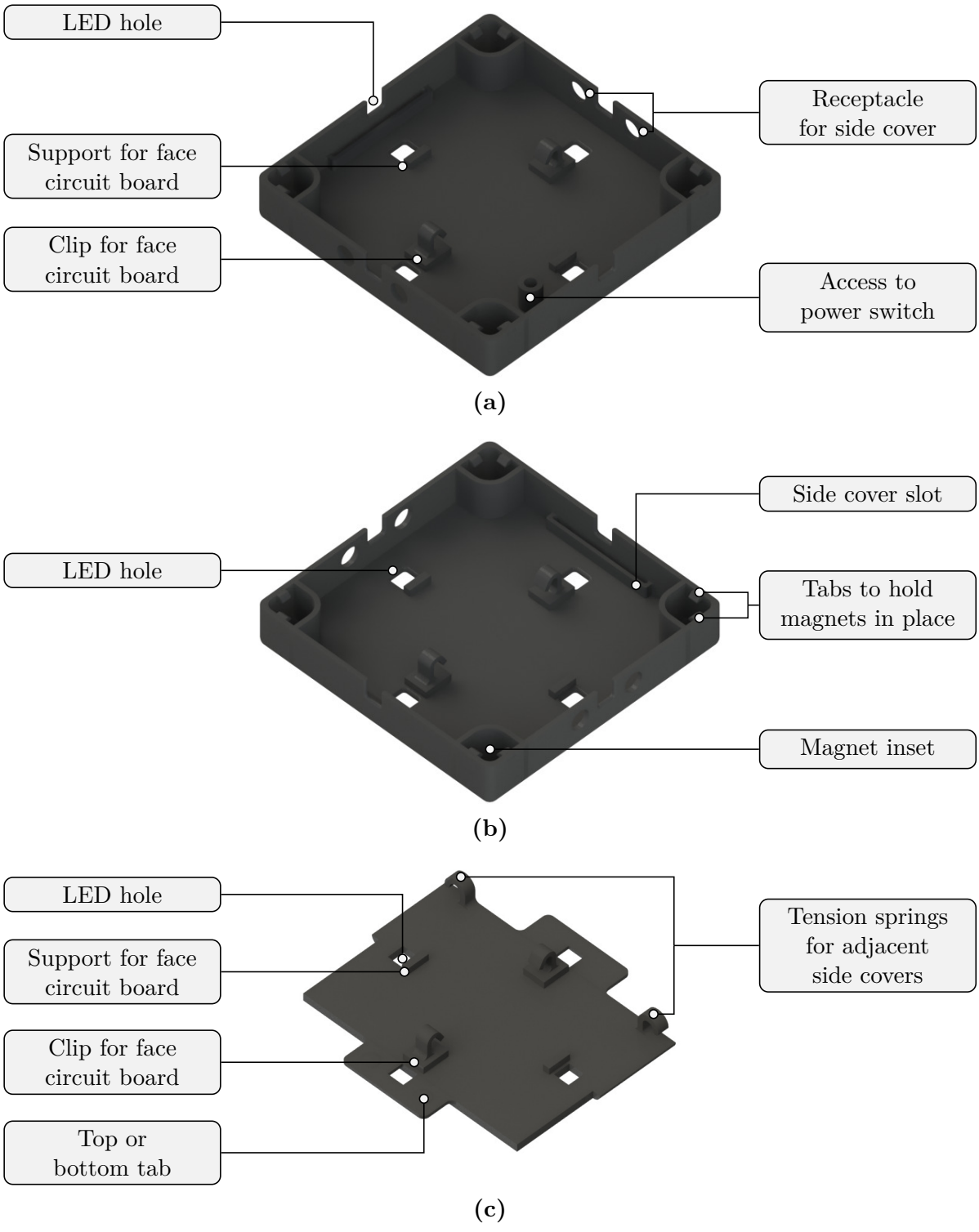


Figure 2.3: Mechanical design of the top (a) and bottom (b) covers and the side cover (c)

A block consists of four side covers, a top cover, and a bottom cover. To assemble a block, a face circuit board is attached to each of these covers using the two small clips. Each side cover and face circuit board assembly is then attached to a central circuit board by connecting the headers on the face circuit board to the receptacles on the central circuit board. A small lithium-ion battery is connected to and placed underneath the central circuit board.

The top and bottom covers both require four spherical magnets to be inserted into the magnet insets. The face circuit boards in the top and bottom covers connect to the central circuit board using a cable. After the spherical magnets have been inserted and the face circuit boards have been connected, the top and bottom covers slide over the side covers to complete the assembly of a block as visually summarized in Figure 2.4. A localizable tag is placed on each cover of a block so that it can be seen by an autonomous robot.

2.3 Software

The microcontroller on the central circuit board contains 32KB of flash memory, 2KB of SRAM, and 1KB of EEPROM. The flash memory is partitioned to include the Optiboot bootloader, which enables reprogramming the microcontroller over USB. The bootloader has been configured with a baud rate of 57600 and runs following a reset of the microcontroller. The microcontroller is programmed using AVRDUDE[9], which automatically resets the microcontroller by pulsing the DTR signal, assuming the Arduino profile is selected.

The firmware for the stigmergic block is written in C++. After the C++ runtime completes its initialization, the microcontroller enters the *main* function, where a singleton instance of a firmware class is created. The constructor for this class initializes several peripherals such as a timer, controllers for the serial ports, and a controller for the I^2C bus. Following initialization of the peripherals, the microcontroller probes each of its ports to determine whether a face circuit board is connected.

For each connected face circuit board, the firmware initializes the NFC controller and the LED driver. Once the firmware has initialized each face circuit board, it enters and remains in a loop, until an interrupt is received from one of the faces circuit boards. This interrupt indicates that a robot is trying to communicate with a block using its NFC interface on one of block's face circuit boards. Upon receiving a message from a robot, a block uses the first byte of this message to configure the color of its LEDs.

This software is relatively simple with respect to the potential functionality of the stigmergic block. For instance, we are currently investigating an implementation of a light-weight pre-emptive operating system for the block. This would enable messages to be sent and received from different faces of a block concurrently, allowing for block-to-block communication inside a structure. Further work based on this block-to-block communication may include adding routing protocols, eventually leading to research into the autonomous construction and maintenance of smart structures.

[9] AVRDUDE: `http://www.nongnu.org/avrdude/`

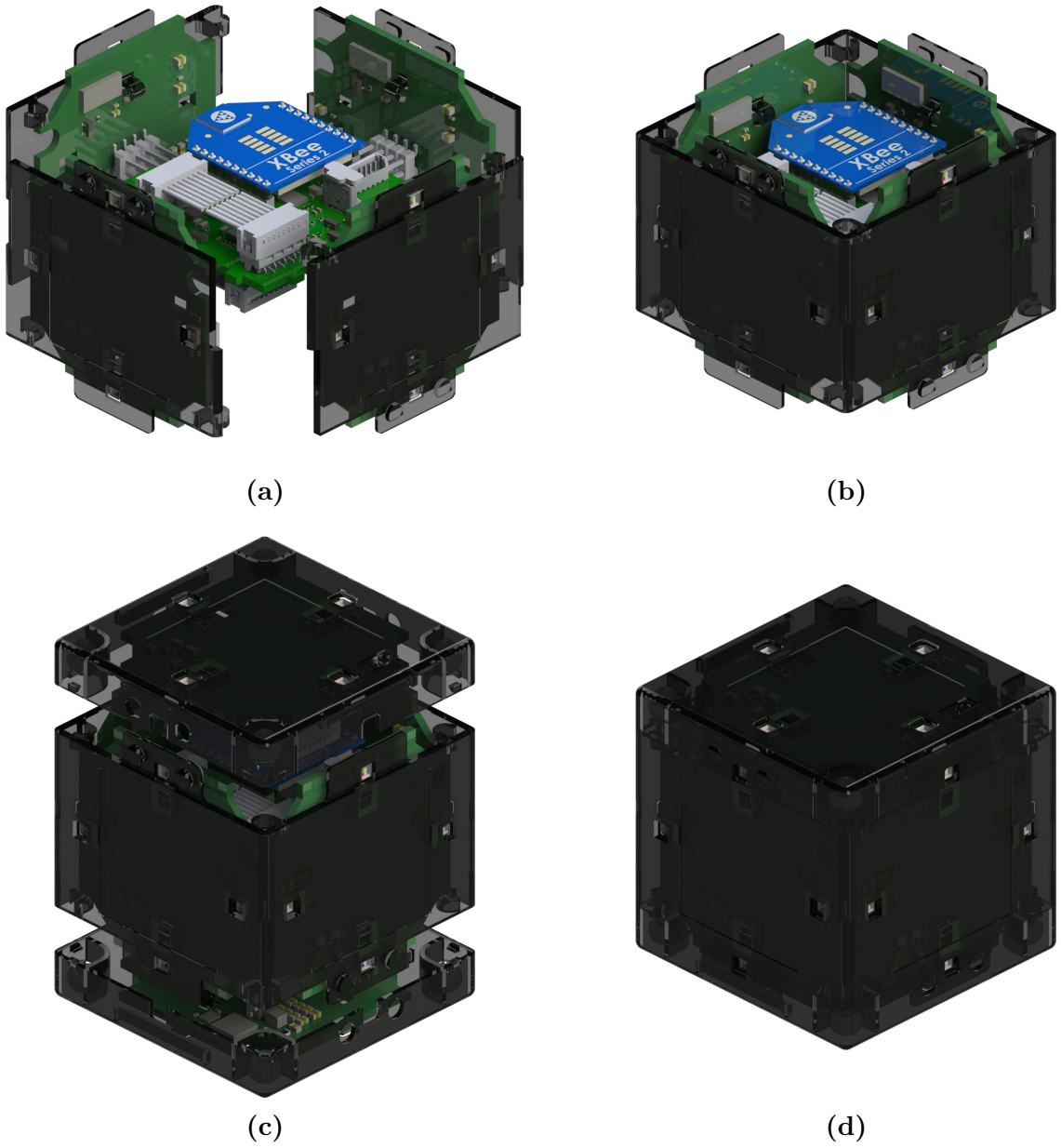
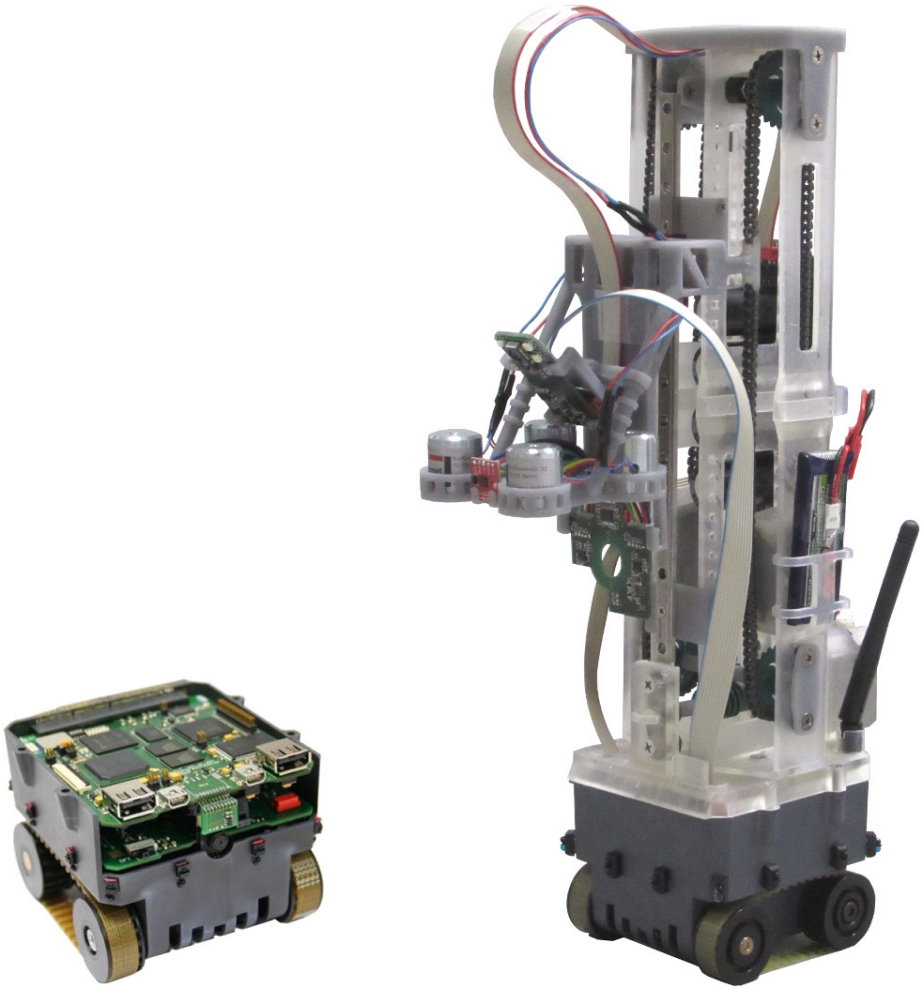


Figure 2.4: The assembly of a stigmergic block: (a) four side covers attach to four face circuit boards following an alternating up down configuration, (b) each side cover and face circuit board assembly connects to the central circuit board, (c) the top and bottom covers attach to two face circuit boards, (d) the top cover and bottom cover, including the attached face circuit boards, slide over the side covers of a block to complete its assembly



(a) **(b)**

Figure 3.1: (a) The BeBot mobile robotics platform (b) the upgraded mobile robotics platform with the manipulator

3 Autonomous Robots

The autonomous robots consist of a mobile robotics platform (an upgraded version of the BeBot [3]) and a manipulator for working with the stigmergic blocks (Figure 3.1). The mobile robotics platform consists of twelve equally-spaced range finders mounted to a molded interconnect device (MID) chassis. The range finders connect to a microcontroller on the chassis, which samples the sensors and provides access to their readings over a serial interface. Two motors mounted to the chassis form a differential drive, allowing the mobile robotics platform to move around its environment. Two circuit boards, which slot into the chassis, are responsible for routing the power, expansion port signals, as well as the sensor and actuator signals to a central microprocessor.

Since the microprocessor used in the original mobile robotics platform was inade-

quate with respect to our computer vision requirements, we have redesigned the two circuit boards around a later generation microprocessor. To reduce development time and manufacturing costs, we have used a Duovero Computer-on-Module (COM) from Gumstix[10].

To enable an autonomous robot to assemble stigmergic blocks into a structure, we have designed a manipulator, which attaches to the top of the mobile robotics platform. The manipulator controls the vertical position of an end-effector, consisting of four semi-permanent electromagnets. These electromagnets couple with the freely-rotating, spherical magnets inside a stigmergic block, holding it in place during transport.

To locate the stigmergic blocks in an environment, the end-effector is equipped with four range finders and a camera. We have mounted the camera at an angle of 45 degrees from the horizontal. This angle provides a compromise between allowing an autonomous robots to see a stigmergic block at a distance as well as when it is nearby. When the end-effector is positioned at its maximum height from the ground (3.5 blocks, or 19.25 centimeters), the camera can see blocks on the ground up to approximately 35 centimeters away from the center of the robot. When the end-effector is positioned at its minimum height from the ground (1 block, or 5.5 centimeters), the blocks can be tracked until they disappear underneath the end-effector.

In the following sections, we provide a detailed overview of the electronics, mechanical design and software of the autonomous robot.

3.1 Electronics

As shown in Figure 3.2, an autonomous robot consists of six interconnected circuit boards. Two of these boards belong to the manipulator, while the other four are part of the mobile robotics platform. Although the camera circuit board is physically connected to the manipulator, we consider it as part of the mobile robotics platform as this is where its power, data, and control signals are routed to and from.

There are two microcontrollers on the power circuit board and one on the manipulator circuit board, all of which communicate with a main microprocessor using a serial interface. This microprocessor runs Linux and is located on the microprocessor circuit board. The microprocessor is capable of reprogramming the microcontrollers on the other circuit boards. We have designed this capability to enable the reprogramming of the attached microcontrollers wirelessly via the microprocessor. This circumvents the need for physical access to the hardware during an upgrade of the software on a large group of robots. The camera and interface circuit boards, as well as the MID chassis, provide access to their sensors and actuators over an I^2C interface. We have not designed these circuit boards to be reprogrammed.

Camera circuit board Computer vision on the autonomous robot is provided using a dedicated circuit board to support an image sensor module from Leopard Imaging

[10]Gumstix: `https://www.gumstix.com/`

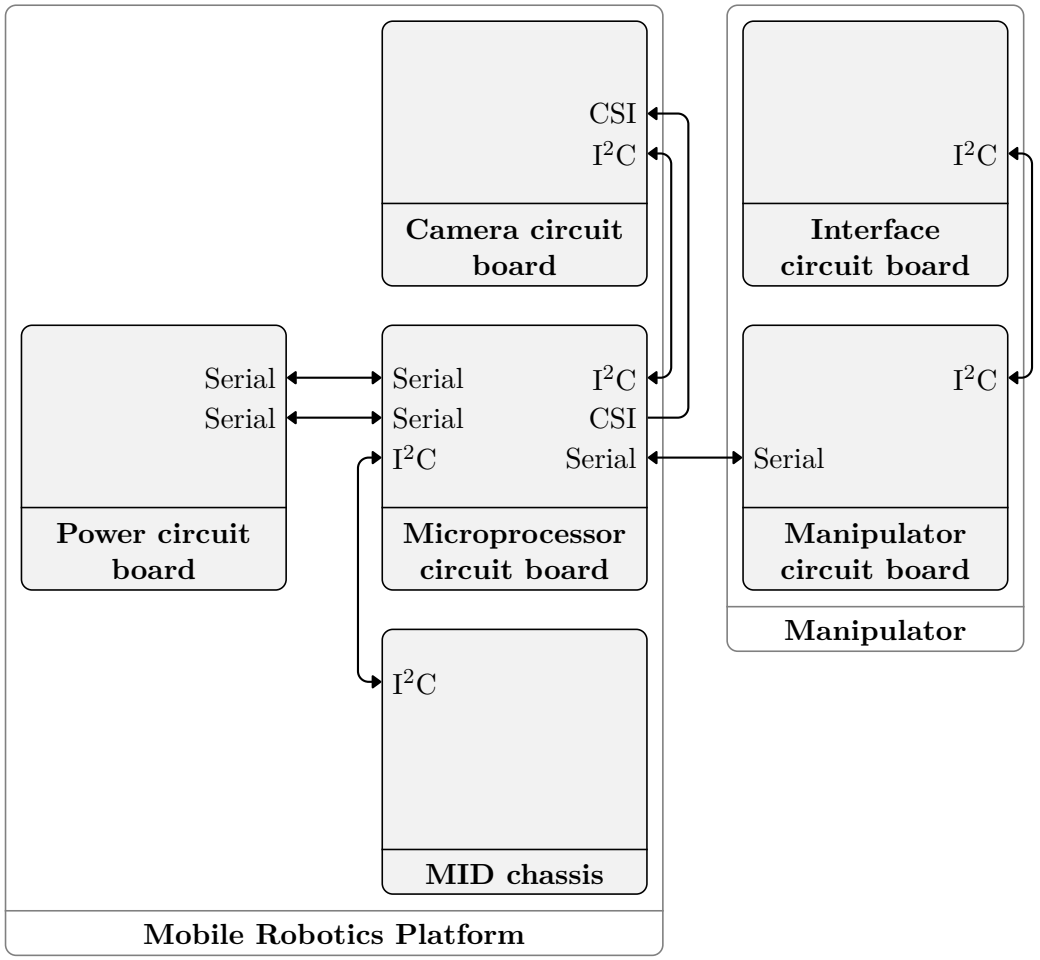
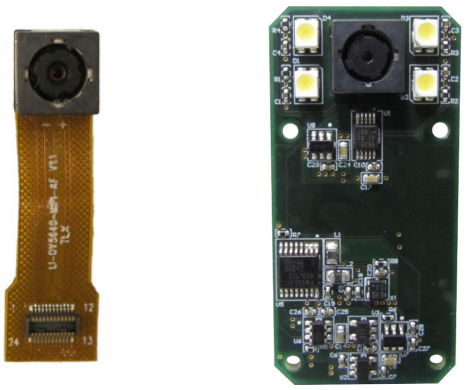


Figure 3.2: Connectivity diagram for the autonomous robot

(Figure 3.3). This image sensor module is based on the OV5640 image sensor from OmniVision.

The OV5640 image sensor requires a clock signal as well as a digital and an analog power supply. We satisfy these requirements using a 24 MHz oscillator, and two low-dropout (LDO) regulators. We have selected LDO regulators to ensure a noise-free power supply for the image sensor. We have placed four white LEDs around the image sensor to optionally enhance the illumination of the captured scene.

The pixel data from the image sensor is routed from the camera circuit board to the microprocessor circuit board using a cable. This cable also provides the camera circuit board with power and the required control signals over an I^2C bus. The I^2C bus connects to the image sensor to control image acquisition, to an LED driver to control the brightness of the four white LEDs, and to a general purpose input/output (GPIO) expander to provide the enable and reset signals for the image sensor and its oscillator and regulators. This level of control is required to correctly power up the image sensor.



(a) (b)

Figure 3.3: Computer vision hardware for the autonomous robot. (a) A module from Leopard Imaging containing the OmniVision OV5640 image sensor. (b) A camera circuit board with an installed module

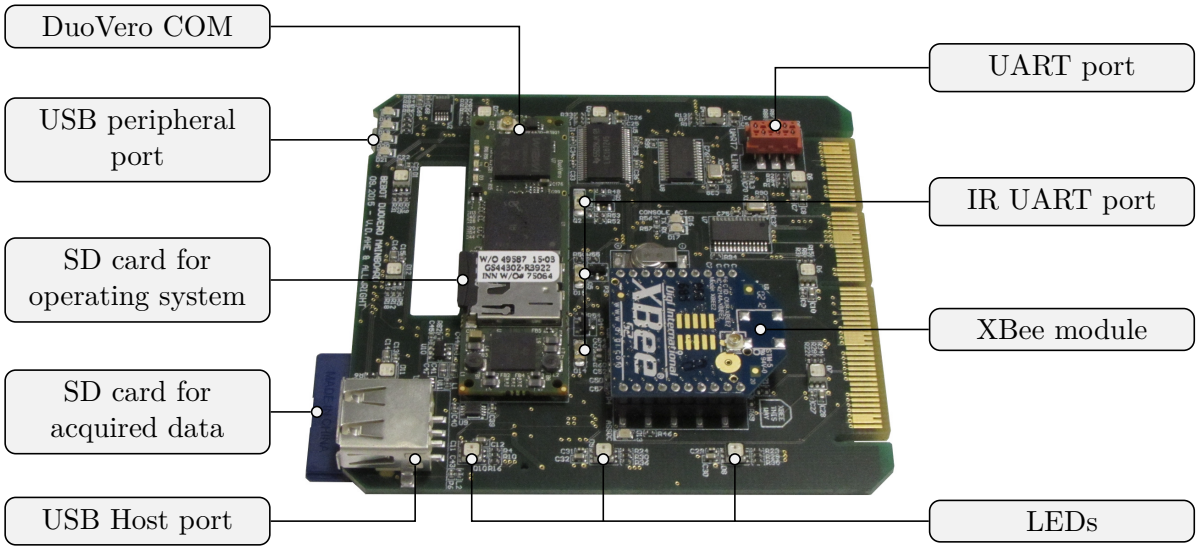


Figure 3.4: The microprocessor circuit board for the mobile robotics platform

Microprocessor circuit board Figure 3.4 shows the DuoVero COM attached to the microprocessor circuit board. The DuoVero COM provides the main microprocessor for an autonomous robot. This microprocessor runs Linux and has two cores, which are clocked at 1 GHz and share 1 GB of memory. The DuoVero COM also provides Bluetooth and facilitates access to a standard wireless network.

The microprocessor on the DuoVero COM provides two camera serial interface (CSI) ports, which can simultaneously capture video. We have routed both of these ports to two custom connectors, which can connect to two camera circuit boards. These connectors are located on the bottom of the microprocessor circuit board near the cut

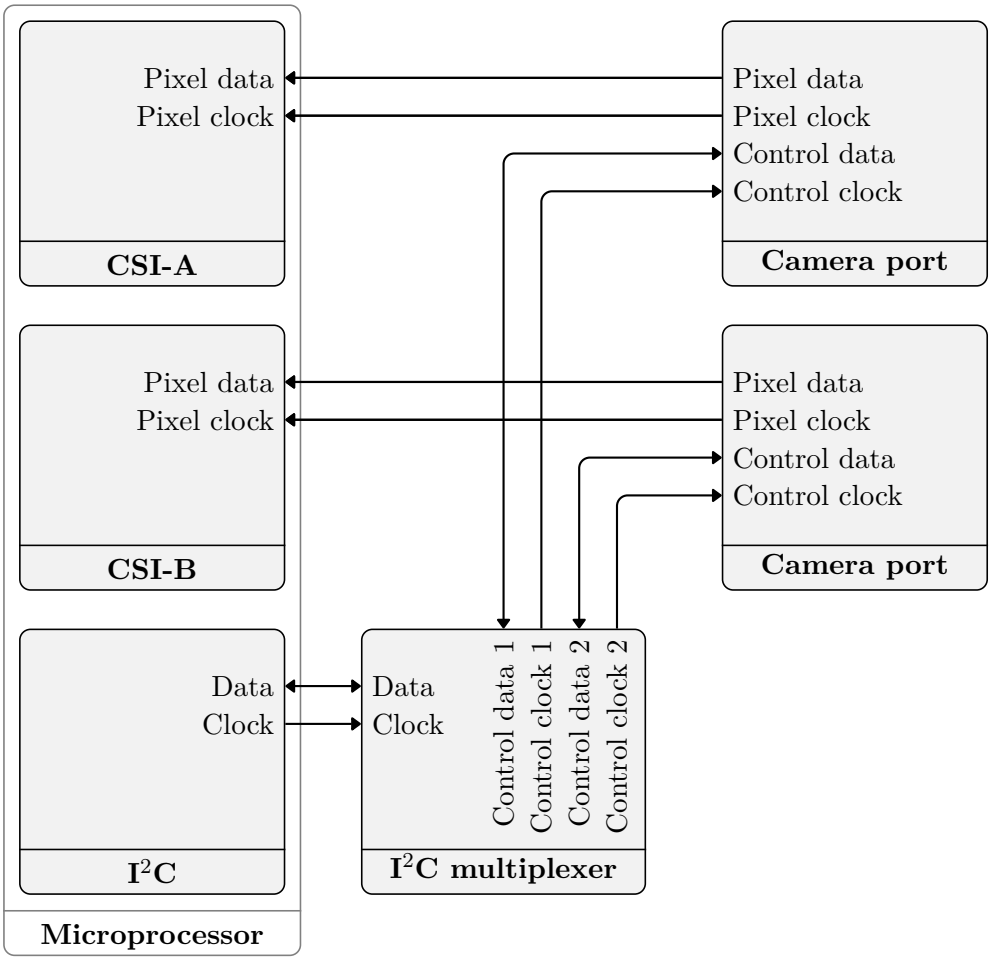


Figure 3.5: Image acquisition connectivity diagram

out on the left-hand side (see Figure 3.4). For a use case involving the capture of video from two identical camera circuit boards, an issue arises due to address conflicts on the I²C bus. Figure 3.5 shows how we have solved this issue by adding a multiplexer to the microprocessor circuit board, segmenting the I²C bus. Capturing video over CSI enables the use of the microprocessor's dedicated image processing hardware. This hardware can capture, scale, and compress the video stream from a connected camera. In contrast, a USB camera requires that most of these operations be performed on the CPU, which consumes resources that an autonomous robot could otherwise use for computer vision.

To store the captured images from an autonomous robot, we have added an SD card reader to the microprocessor circuit board. In addition, a standard USB host port is provided, to which any standard USB device can be connected.

To ease development, an autonomous robot can be connected to a PC via its micro-USB port. This port is routed to an integrated USB hub, which provides a developer with low-level access to a robot's bootloader (via an onboard USB-to-serial converter) and high-level access to the robot's operating system by emulating an Ethernet connection

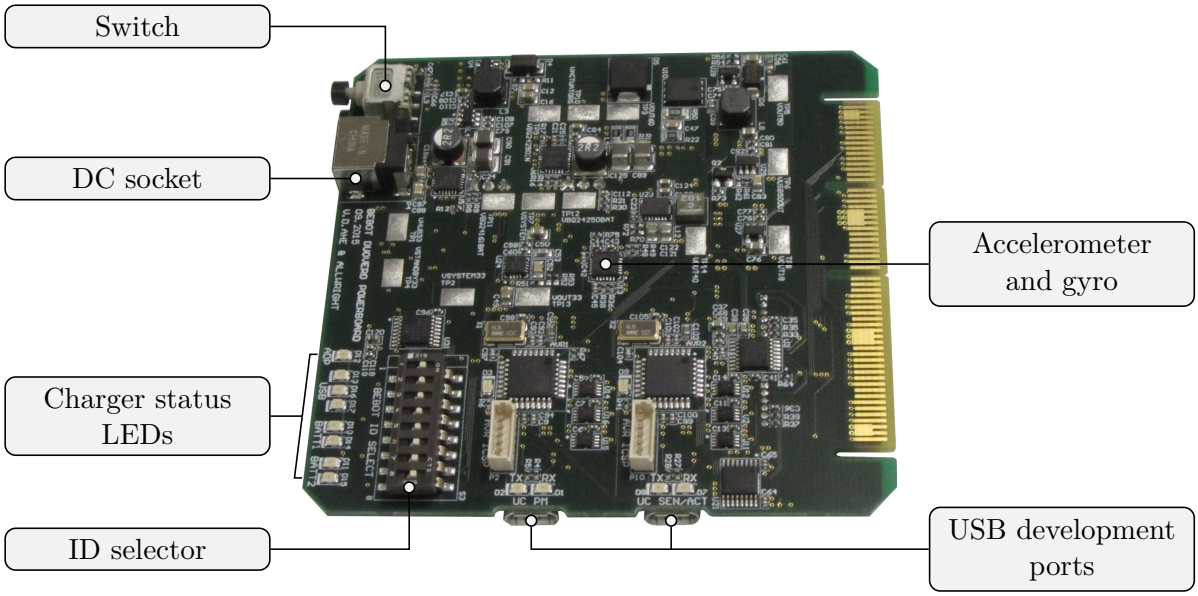


Figure 3.6: The power circuit board for the mobile robotics platform

over USB On-The-Go (OTG). The integrated USB hub is compliant with the USB battery charging specification, enabling the robot to safely draw current from the USB connection in order to run its system, charge its batteries, or both.

The DuoVero COM on the microprocessor circuit board provides two serial ports. The first serial port is used to access the robot's bootloader and the second serial port is connected to a socket for a low-power wireless module[11]. In our design, we require four further serial connections to communicate with the two microcontrollers on the power circuit board, the microcontroller on the manipulator circuit board, and one additional serial connection for an infrared interface, which is used to maintain backward compatibility with the modules described in [4]. To satisfy this requirement, we include two I^2C to serial bridges in our design, which provide the four serial connections.

For debugging and inter-robot communication, twelve multi-color LEDs are evenly spaced around the perimeter of the microprocessor circuit board.

Power circuit board The power circuit board hosts two systems: the sensor-actuator system and the power management system. The sensor-actuator system provides a differential drive for the mobile robotics platform. This system contains a microcontroller, which implements a closed-loop controller for the left and right wheels. Embedded shaft encoders in the motors enable the microcontroller to measure changes in the position of the wheels. The target velocity for the closed-loop controller is set by the microprocessor using a serial interface.

[11] Xbee Wireless Modules: `https://www.digi.com/lp/xbee`

The closed-loop controller for the wheels has an update period of 16.3 milliseconds. During this period, the rotation of the two wheels is measured using an interrupt routine, which is triggered on the rising and falling edges of the shaft encoder signals. A timer overflow interrupt fires at the end of this period, triggering an update of the closed-loop controller. The closed-loop controller then calculates two output duty cycles to be applied to the motors over the next period. These duty cycles are used to configure the first timer on the microcontroller to generate two pulse width modulation (PWM) waveforms. These waveforms are routed to a motor driver, which is connected to the left and right motors.

The closed-loop controller consists of two PID controllers for the left and right wheels. The tuning of these controllers is prodominantly integral, due to issues with the mechanical design of the original hardware. As the wheels are directly attached to motor shafts, the entire weight of the autonomous robot creates a bending moment along the shafts of the motors, probably interfering with the operation of the internal planetary gearboxes in the motors. This interference results in unpredictable friction and occasional jamming. We were only able to partially mitigate these issues by tuning the PID controllers.

In addition to the differential drive for the mobile robotics platform, the sensor-actuator system includes a digital gyroscope-accelerometer sensor. The readings of this sensor are made available to the microprocessor over a serial interface.

The power management system is responsible for routing power and for recharging the batteries in the mobile robotics platform. The power management is broken down into two domains: the system power domain and the actuator power domain. Both of these power domains have their own battery and power management IC (PMIC). External power can be applied to the mobile robotics platform using either the standard 5.5/2.1 millimeter power jack on the power circuit board or the micro-USB connector on the microprocessor circuit board. As shown in Figure 3.7, the external power inputs are connected to the system power management IC, which routes power to the actuator power management IC.

The software controlling the power management system is implemented on a microcontroller. This software configures the integrated USB hub on the microprocessor circuit board and reads the result from the USB hub's battery charger detection circuitry. The software on the microcontroller allocates the power to the mobile robotics platform according to the following prioritized list:

1. system power (if switched on)
2. actuator power (if switched on)
3. system battery (if battery is low)
4. actuator battery (if battery is low)

For each above use of power, the microcontroller subtracts the required amount of power for that use from the remaining available power, which is initially calculated from the input power to the system. As the batteries can be recharged at different rates, the software sets the recharge rate with respect to the remaining available power.

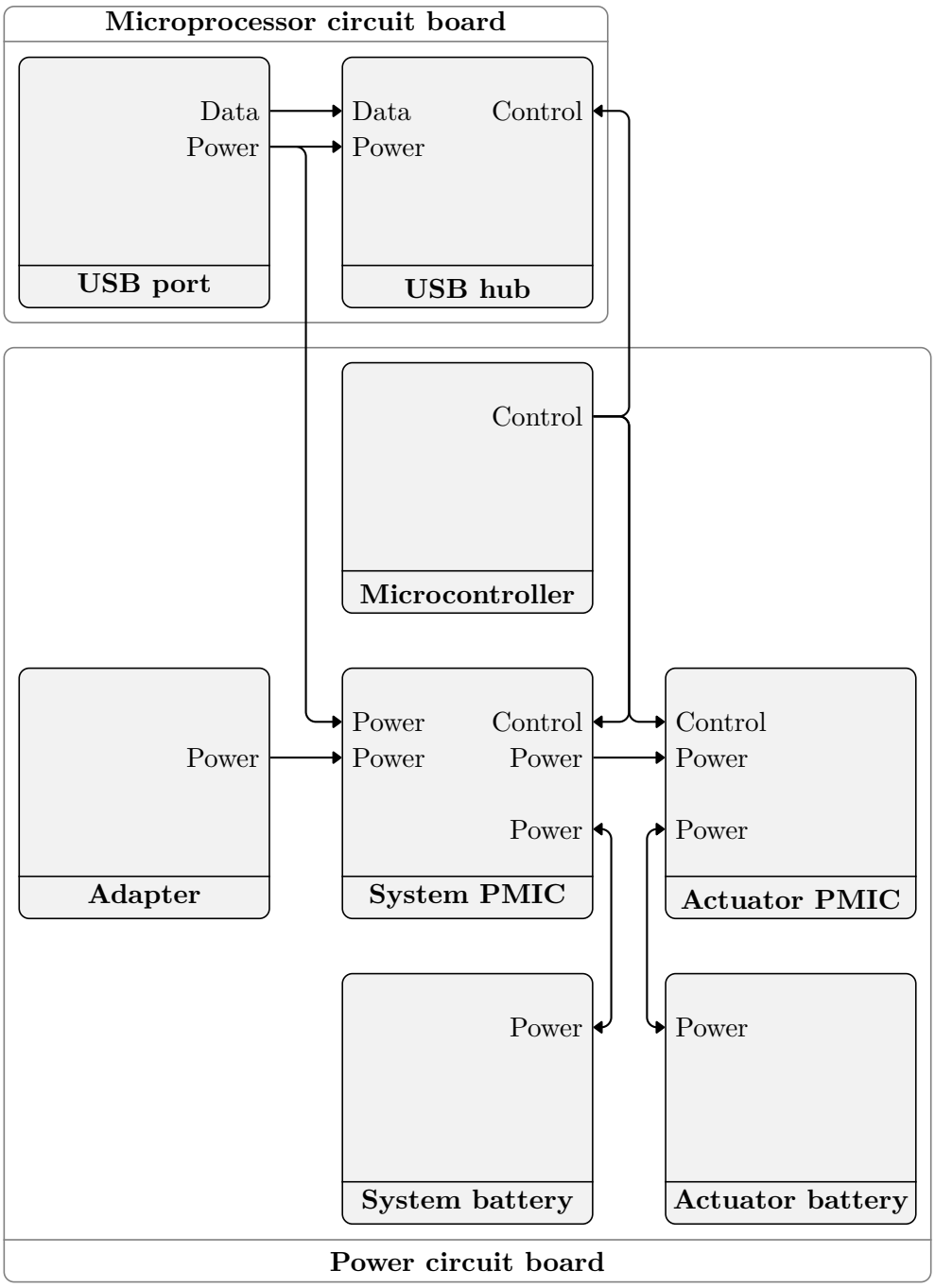


Figure 3.7: Power management system for the mobile robotics platform

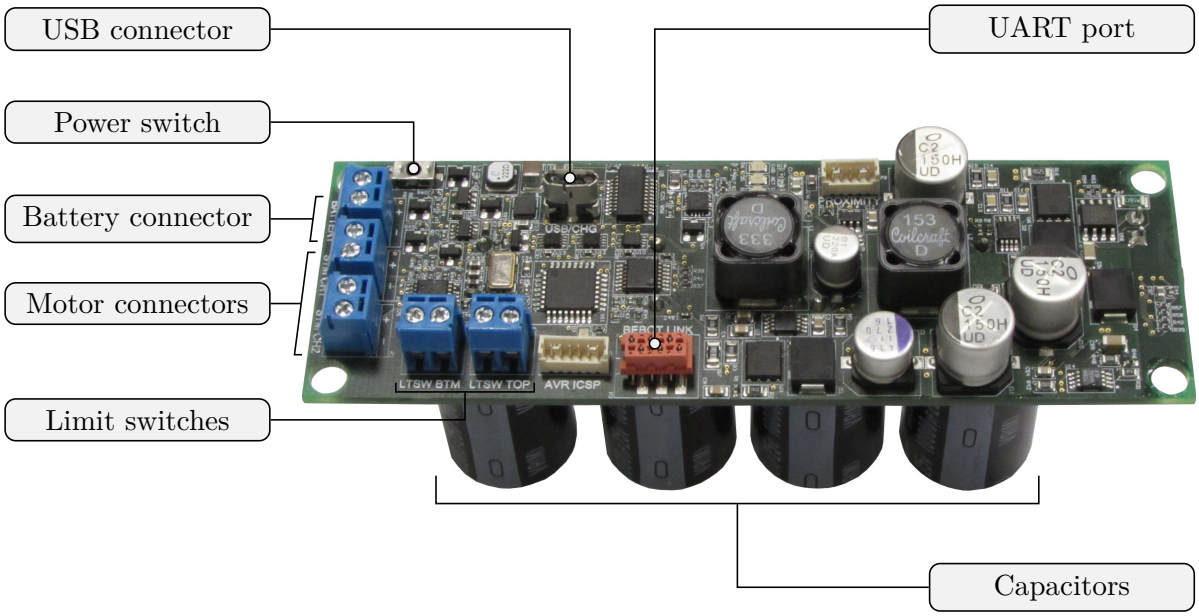


Figure 3.8: Manipulator circuit board

Manipulator The manipulator adjusts the height of the stigmergic blocks by raising and lowering its end-effector. The height of the end-effector is controlled using a stepper motor and is constrained by two limit switches at the top and bottom of the manipulator. These limit switches trigger as the end-effector starts to move out of range. Four semi-permanent electromagnets are located in the end-effector, which couple with the spherical magnets in a stigmergic block. Depending on the direction of a current applied to the semi-permanent electromagnets, the magnet field can be either strengthened or weakened. This current is generated by precharging four 6.8 millifarad capacitors to 25 volts. The direction of the current is controlled using a H-bridge. An autonomous robot strengthens the magnetic field during block attachment. This strengthening of the field improves the alignment of the stigmergic block with the end-effector prior to the attachment. An autonomous robot weakens the magnetic field in order to detach a block and assemble it into a structure.

In addition, the electronics for the manipulator consists of two circuit boards. A main circuit board (Figure 3.8) and an interface circuit board (Figure 3.9).

The main circuit board contains a microcontroller, which executes the manipulator's software and communicates with the microprocessor on the mobile robotics platform using a serial connection. This serial connection is multiplexed with a USB-to-serial converter. When a USB cable is attached, the serial connection is rerouted over the USB connection to a development PC, which can be used for debugging and upgrading the manipulator's software. The USB-to-serial converter also implements battery charger detection, which configures a power management IC. The manipulator contains its own battery and is charged over the USB connection.

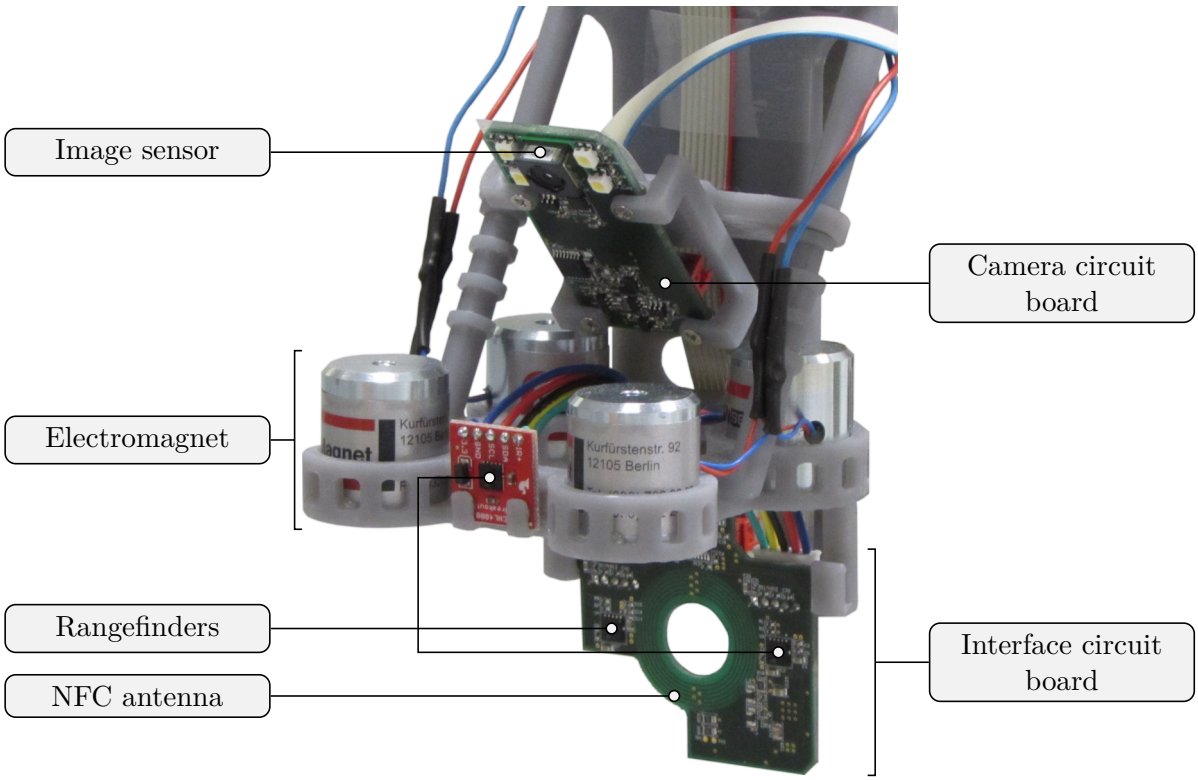


Figure 3.9: Interface circuit board of the manipulator attached to the end-effector

The interface circuit board contains an NFC transceiver for communicating with a stigmergic block. Two rangefinders are mounted directly to the interface circuit board, which an autonomous robot uses during alignment with a block or with a structure. The interface circuit board also provides two connectors for two additional range finders which are connected to the end-effector.

The software on the microcontroller samples the rangefinders, implements a controller for the NFC transceiver, and regulates the precharging of the capacitors as well as the configuration of the semi-permanent electromagnet H-bridge. The software on the microcontroller also implements an open-loop controller for the height of the end-effector. This controller performs self-calibration of the end-effector, regulates the position of the end-effector, and monitors the state of the limit switches.

3.2 Mechanical Design

The autonomous robot consists of the mobile robotics platform and a manipulator, which is mounted on top of the platform. Two motors in the mobile robotics platform constitute a differential drive, allowing the robot to move around its environment. The footprint of the mobile robotics platform is a square with a side length of 9 centimeters. The height

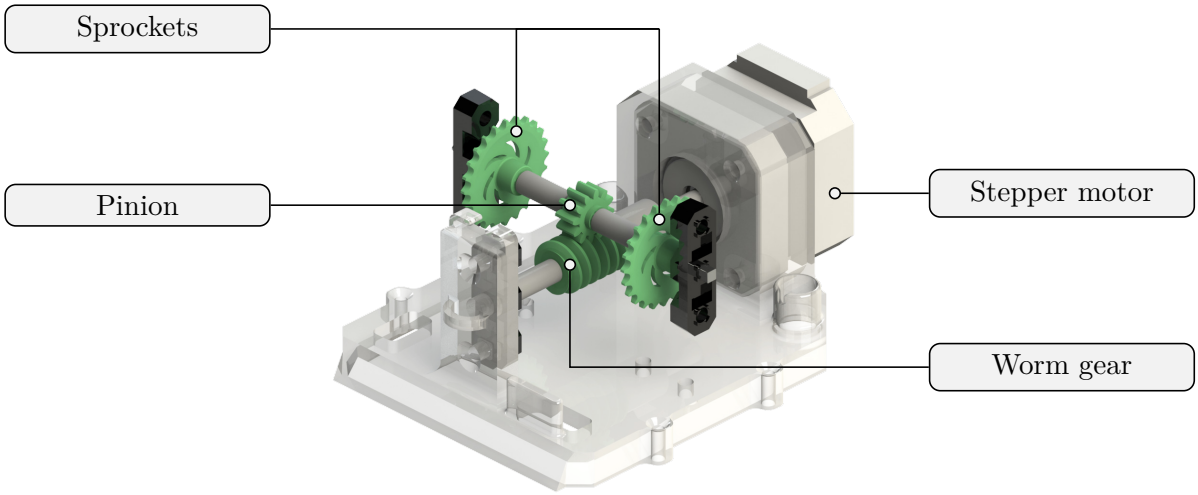


Figure 3.10: Component diagram for the manipulator base

of the platform is 7 centimeters.

The manipulator is 30 centimeters tall, and once mounted on top of the mobile robotics platform, gives the autonomous robot an overall height of 37 centimeters. Figure 3.10 shows the base of the manipulator, which is printed from a clear photopolymer resin using stereolithography. A stepper motor is attached to the base using four screws. The motor's shaft is supported by a bearing and drives a worm gear. This worm gear interfaces a pinion, which rotates the lower shaft, driving two sprockets. These sprockets are connected to chains which provide the upwards and downwards motion required by the end-effector. The shafts, bearings, worm gear, pinion, and sprockets are all off-the-shelf components, which have been sourced from Vex Robotics[12].

Figure 3.11 shows two chains running from the bottom sprockets to two upper sprockets, which are suspended by two upper shafts and four bearings. These chains attach directly to the end-effector to change the end-effector's height. To balance the load on the chain, two lead counterweights attach to the chain, opposite the end-effector. Together with the weight from the stepper motor, the weight of the counterweights balances the weight of the electromagnets at the front of the autonomous robot.

The electromagnets, counterweights, sprockets and bearings are off-the-shelf components. The remaining structural components have been printed using either a gray or a clear photopolymer resin using stereolithography. Figure 3.12 shows an example part from the manipulator, which we have printed using this technique. The part is the top of the manipulator structure, which aligns the manipulator's columns and supports the bearings for the upper shafts. To prepare this part for use, the support material must be removed. To improve functionality and aesthetics, it is necessary to finish the parts with sandpaper and a polish for plastic surfaces.

[12]Vex Robotics: `http://www.vexrobotics.com/`

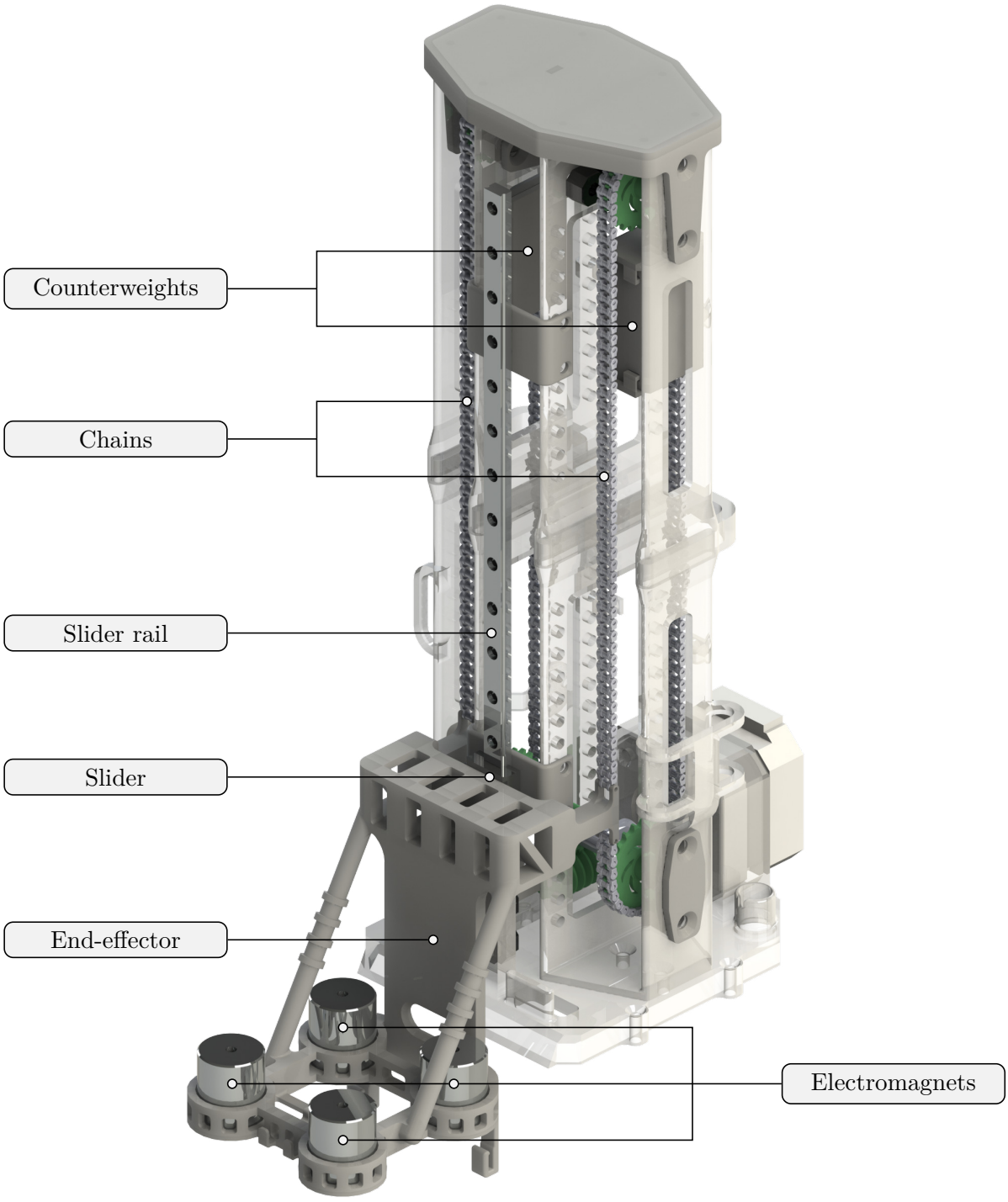


Figure 3.11: Component diagram for the manipulator

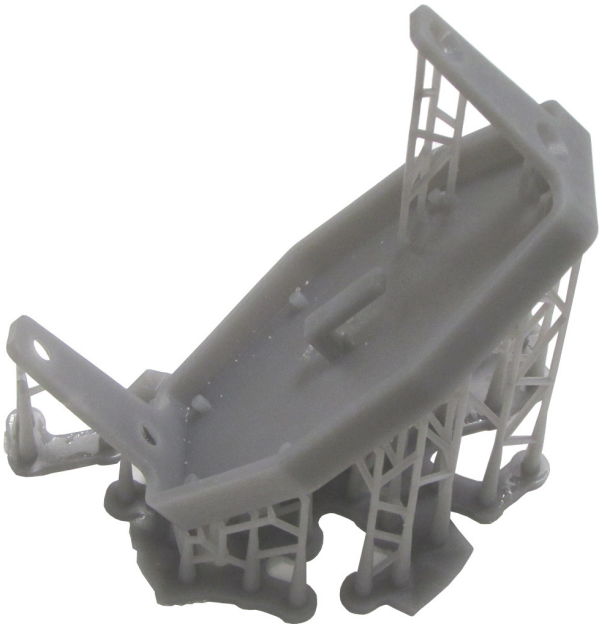


Figure 3.12: An example of a component for the manipulator printed using stereolithography

3.3 Software

Operating system The microprocessor in the mobile robotics platform runs a custom variant of the Linux operating system. This custom variant of Linux is downloaded, configured, and compiled by the Yocto build system[13]. The build system uses *recipes*, which describe the numerous tasks required to prepare a bootable *image* for an embedded system. These tasks may include fetching software from a version control system, applying local patches, as well as compiling and installing the software into the target root file system.

The image for the mobile robotics platform is based on the Gumstix console image[14], which provides a basic configuration of the Linux operating system for the microprocessor. This configuration includes a shell, utilities, and tools for networking and system configuration. We have enhanced this image to support the hardware on the mobile robotics platform by configuring the Linux kernel and adding additional software packages.

We have selected the hardware for the mobile robotics platform with respect to the availability of drivers in the Linux kernel mainline. These drivers are considered to be stable and are well maintained by the Linux community. Furthermore, we have selected hardware for which device tree bindings already exist[15]. This choice of hardware significantly simplifies the process of configuring Linux for the mobile robotics platform.

We have, however, encountered a bug in the Linux kernel, which prevents the clocks

[13]Yocto Project: `https://www.yoctoproject.org/`

[14]Gumstix Developer Center: `http://gumstix.org/`

[15]The Devicetree Specification: `https://www.devicetree.org/`

of peripheral devices from being detected in the device tree. This bug is fixed in a later version of the Linux kernel (4.1), which, however, had an issue with the power management of the microprocessor's imaging subsystem. During the configuration of the Linux kernel for the mobile robotics platform, we also found an issue with enable signals for peripheral clocks connected to I^2C GPIO expanders. This configuration generates kernel warnings due to the latencies involved with enabling a clock over an I^2C bus. To resolve these issues in a timely manner, we have used an older version of the Linux kernel (3.17) that supports the imaging subsystem. To work around the clock issues, we have patched the drivers to enable their clocks using the Linux GPIO interface and hard coded the respective clock frequencies into the drivers.

Although the driver used for the microprocessor's imaging subsystem is in the Linux kernel mainline, it is part of the staging directory and its quality is not guaranteed. Furthermore, the driver for the robot's camera is out-of-tree and is unmaintained by the Linux community. These drivers required patching before they would work on the mobile robotics platform.

We enhanced the Gumstix console image with additional software packages to support our application. For example, we have included tools for capturing and working with images from the robot camera such as media-ctl[16], yavta[17], and OpenCV[18]. We have added a recipe to compile and install the detector from the AprilTags visual fiducial system as a shared library (see [2]). We have also created a test application called *blocktracker*, an executable which configures the autonomous robot and runs test routines. We have extended the blocktracker test application to implement the hardware experiments presented in this work.

Blocktracker The behavior of the autonomous robot is implemented by the blocktracker executable, which is written in C++ and built using CMake[19]. The executable has four main components: a packet control interface, an image processing pipeline, a finite state machine, and a control loop.

The microcontrollers on the manipulator and power circuit boards communicate with the microprocessor using a packet control interface. We have designed this interface to support multi-byte commands. The packet control interface also checksums the commands and validates their length. Figure 3.13 shows an example command, which queries the battery voltage on a remote microcontroller. A valid command always starts with a two-byte preamble and ends with a two-byte postamble. We have selected the values of these bytes due to their visibility on an oscilloscope. As the example command has no arguments, its length field is zero. For commands which do have arguments, the length field is nonzero and the bytes for the arguments are inserted between the length and checksum fields. In this case, the checksum field represents the summation of the bytes of the arguments. Incoming bytes are stored in a buffer and searched for

[16]Media-ctl: `https://git.linuxtv.org/v4l-utils.git/tree/utils/media-ctl`

[17]Yavta: `http://git.ideasonboard.org/yavta.git/tree`

[18]OpenCV: `http://opencv.org/`

[19]CMake: `https://cmake.org/`

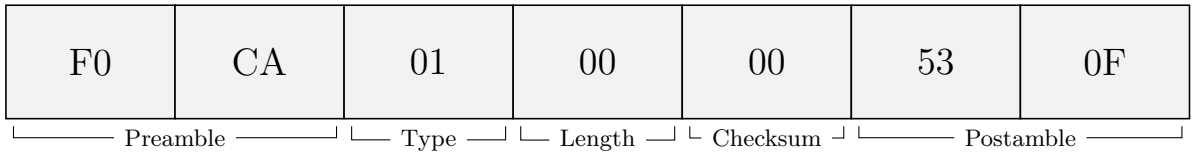


Figure 3.13: An example command for the packet control interface

valid commands using a specialized state machine. If the buffer overflows or an invalid command is detected, the state machine searches for the next preamble, flushing out any data found before it.

The image processing pipeline is broken down into an asynchronous component and a synchronous component. The implementation of the asynchronous component was motivated due to an earlier implementation, where the microprocessor wasted significant processing time, waiting for the microcontrollers on the manipulator and power circuit boards to respond to commands. The asynchronous component of the pipeline enables the microprocessor to capture and process images from the camera while waiting for the microcontrollers to respond. The asynchronous component of the pipeline is built on top of the concurrency extensions to the C++ standard library[20]. The implementation of the asynchronous component defines an operation class, which contains an operation method, a management thread, a queue of image buffers to be processed, and a pointer to the next operation. When an operation is enabled, its management thread inspects the queue of image buffers every five milliseconds. If an image buffer is found in the queue, the queue is temporarily locked and the image buffer is extracted for processing. Once processing is complete, the operation attempts to lock the queue of the next operation as defined by its next pointer. Once this queue is locked, the operation moves the processed buffer into the next operations queue.

The operation class is specialized to perform the following functions: (i) to capture a frame from the camera, (ii) to stream a frame over a wireless network, (iii) to save a frame to local memory, (iv) to detect the stigmergic blocks in a frame, and (v) to annotate a frame with the output from the detection operation. Upon initialization of the pipeline, we create four image buffers and enqueue them inside the capture operation. The operations are connected in a loop so that the image buffers are recycled and dynamic memory allocation is not required while the pipeline is running.

The stigmergic block detection operation uses the detector from the AprilTags visual fiducial system to find the tags on the stigmergic blocks. This operation also samples the colors of the LEDs on a stigmergic block, which are used for block-to-robot communication. As the images from the microprocessor's imaging subsystem are in the YUV format, we have selected four LED colors with respect to the four quadrants of the UV color space (Figure 3.14). This selection of colors from the UV color space avoids the requirement of performing a color space conversion.

[20]Concurrency extensions in C++: `https://isocpp.org/wiki/faq/cpp11-library-concurrency`

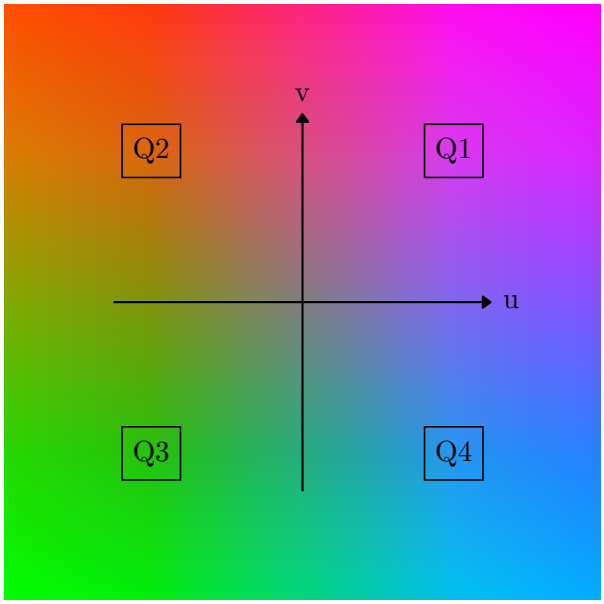


Figure 3.14: The YUV color space with Y = 0.5, showing the four colors used to represent different types of blocks

The stigmergic block detection operation maintains a queue of its detections from the processed frames. This queue is lockable and accessible from the synchronous component of the image processing pipeline, which is executed in the control loop. The synchronous component of the image processing pipeline tracks stigmergic blocks using the Hungarian algorithm with a modified cost matrix, which accommodates new and lost blocks [5]. The tracked blocks are clustered into structures by comparing the distance between any two blocks with a threshold. The results of the image processing pipeline are then available to the finite state machine.

We have developed a state machine library for implementing the behavior of an autonomous robot. The motivation for developing this library (as opposed to using a pre-existing library) was to take advantage of the features in C++11 to create a state machine library that supported reusable sub-states while being compact and easy to read. An example instantiation of a state machine using this library is shown in Listing 3.1. This example demonstrates how classes inherited from the `CState` class can be composed and customized to implement arbitrary behavior. For instance, the class `CStatePrint` inherits from `CState` to define a state that writes the contents of a `std::string` to a `std::ostream`. Line 25 of the example shows how the class `CStatePrint` is configured using its constructor via the templated `AddState` method. The `AddState` method uses the perfect forwarding and variadic template mechanisms of C++11 to automatically insert the parent pointer as the second argument to the `CStatePrint` constructor. The class `CStateFooBar` demonstrates how the `CStatePrint` class is composed using the initialization list and how the transitions are defined in the constructor body to create the finite state machine shown in Figure 3.15, which writes the string "foobar" to `std::cout` and exits.

```
#include <iostream>

#include "state.h"

class CStatePrint : public CState {
public:
   CStatePrint(const std::string& str_id,
               CState* pc_parent,
               std::ostream& c_device,
               const std::string& str_data) :
      /* init the base class id and function with a lambda */
      CState(str_id, pc_parent, [this, &c_device] {
         c_device << m_strData;
      }),
      m_strData(str_data) {
   }
   std::string m_strData;
};

class CStateFooBar : public CState {
public:
   CStateFooBar(const std::string& str_id, CState* pc_parent = nullptr) :
      CState(str_id, pc_parent, nullptr, CState::TVector {
         /* add states */
         AddState<CStatePrint>("print_foo", std::cout, "foo"),
         AddState<CState>("print_bar", nullptr, CState::TVector {
            /* add sub-states */
            AddState<CStatePrint>("print_b", std::cout, "b"),
            AddState<CStatePrint>("print_a", std::cout, "a"),
            AddState<CStatePrint>("print_r", std::cout, "r"),
         }),
      }) {
      /* declare state transitions */
      AddTransition("print_foo","print_bar");
      AddExitTransition("print_bar");
      /* declare sub-state transitions */
      GetState("print_bar").AddTransition("print_b","print_a");
      GetState("print_bar").AddTransition("print_a","print_r");
      GetState("print_bar").AddExitTransition("print_r");
   }
};

int main(int argc, char* args[]) {
   /* instansiate state machine */
   CStateFooBar cStateFooBar("fsm");
   /* run until exit transition */
   for(;;) {
      if(cStateFooBar.Step() != false) {
         break;
      }
   }
   return 0;
}
```

Listing 3.1: Example instantiation of a finite state machine that writes the string “foobar” to `std::cout` and exits

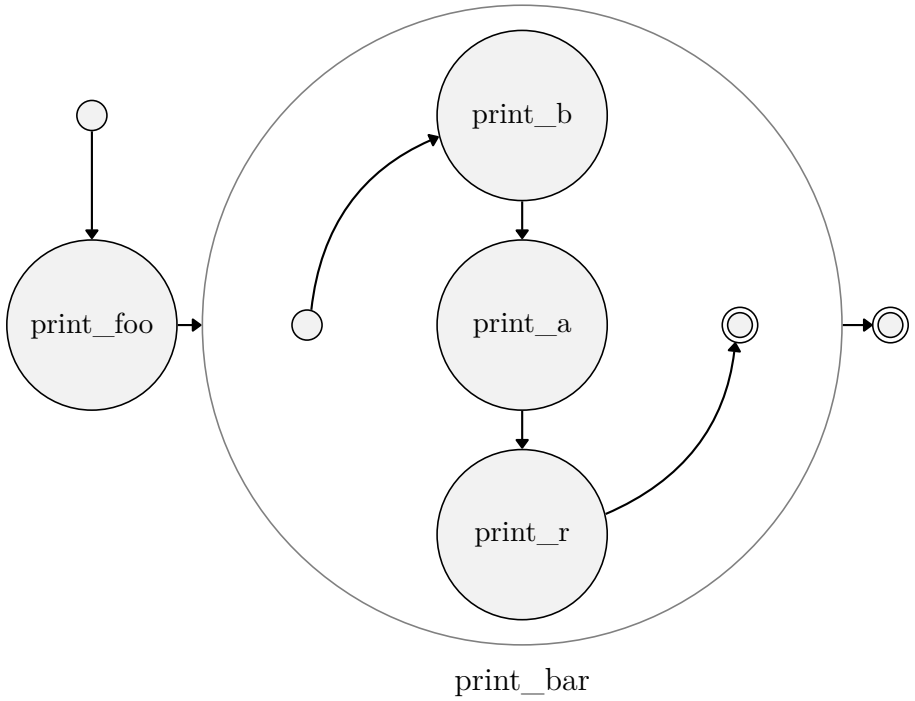


Figure 3.15: The finite state machine `CStateFooBar` as produced by the code in Listing 3.1

The blocktracker executable starts with an initialization routine, which connects to the remote microcontrollers on the power and manipulator circuit boards and checks if they are responding to commands using the packet control interface. The initialization routine then configures the microprocessors imaging subsystem as well as the camera, before initializing the image processing pipeline.

The blocktracker executable then enters the `exec` method. This method starts by enabling the actuator power domain and the differential drive system. The method then requests the manipulator to perform its self-calibration routine. After the calibration is complete, the `exec` method enters the control loop, which samples an autonomous robot's sensors, steps its behavioral state machine, and updates its actuators. The control loop continues until the behavioral state machine exits. The image processing pipeline is the largest bottleneck in the control loop and limits the update period to approximately 160 milliseconds in good lighting conditions. To keep the update period for the high-level closed-loop controllers constant, we lengthen the update period so that it always lasts 200 milliseconds.

Figure 3.16 shows the base behavioral state machine for an autonomous robot. The state machine starts with an autonomous robot searching its environment for unused blocks. Once an unused block is found, the robot picks it up and begins to search the environment for a partially-built structure. When such a structure is found, the robot inspects it to determine if a pattern of blocks with an associated construction action can be found. If such a pattern is found, the robot performs the construction action.

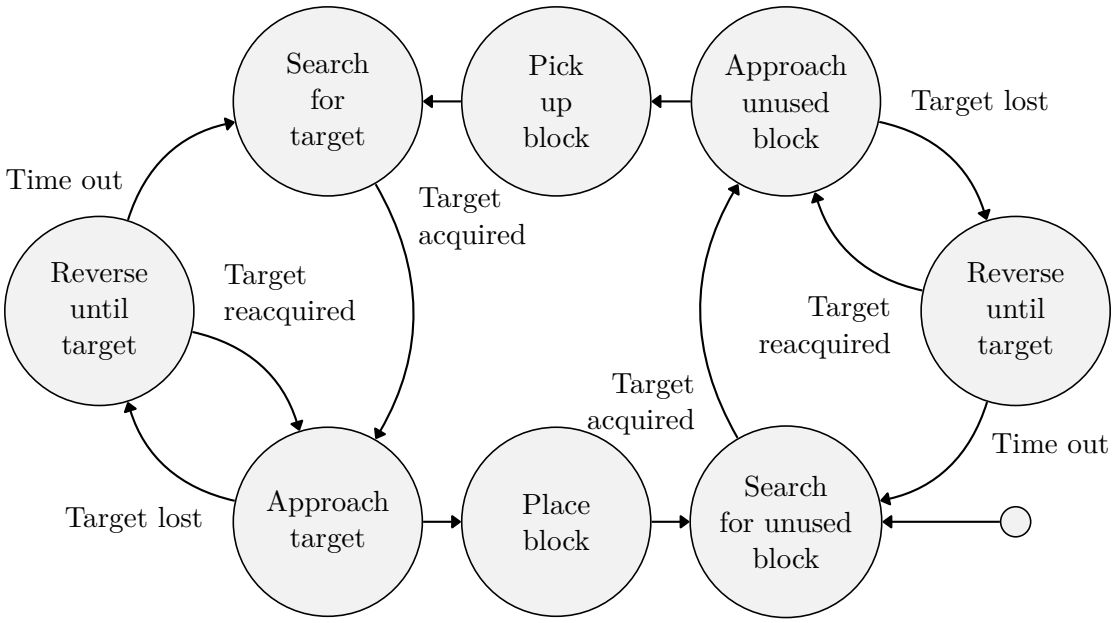


Figure 3.16: The base behavioral state machine for an autonomous robot

Otherwise, the robot continues searching for another partially-built structure. Patterns are defined by matching a set of conditions, which are expressed in terms of the structural arrangement of the stigmergic blocks and their LED markings.

4 Conclusion

This document has summarized the electronics, mechanical design, and software of the stigmergic block and the autonomous robot used in our multi-robot construction system. For demonstrations of how this hardware is used to enable multi-robot construction, we refer the reader to the main article [1] and its accompanying videos, which can be found online[21].

5 References

[1] Michael Allwright, Navneet Bhalla, Carlo Pinciroli, and Marco Dorigo. "Towards Autonomous Construction using Stigmergic Blocks". Technical report TR/IRIDIA/2017-003. Brussels, Belgium: IRIDIA, Université Libre de Bruxelles, 2017.
URL http://iridia.ulb.ac.be/IridiaTrSeries/index.php

[2] Edwin Olson. "AprilTag: A Robust and Flexible Visual Fiducial System". In: *2011 IEEE International Conference on Robotics and Automation (ICRA 2011)*. IEEE Press, 2011, pages 3400–3407.
DOI https://dx.doi.org/10.1109/icra.2011.5979561

[3] Stefan Herbrechtsmeier, Ulf Witkowski, and Ulrich Rückert. "BeBot: A Modular Mobile Miniature Robot Platform Supporting Hardware Reconfiguration and Multi-standard Communication". In: *Progress in Robotics*. Volume 44. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2009, pages 346–356.
DOI https://dx.doi.org/10.1007/978-3-642-03986-7_40

[4] Jürgen Gausemeier, Thomas Schierbaum, Roman Dumitrescu, Stefan Herbrechtsmeier, and Alexander Jungmann. "Miniature robot BeBot: Mechatronic test platform for self-x properties". In: *Proceedings of the Ninth International Conference on Industrial Informatics (INDIN 2011)*. IEEE Press, 2011, pages 451–456.
DOI https://dx.doi.org/10.1109/indin.2011.6034921

[5] Felix Lütteke, Xu Zhang, and Jörg Franke. "Implementation of the Hungarian Method for Object Tracking on a Camera Monitored Transportation System". In: *Proceedings of the Seventh German Conference on Robotics (ROBOTIK 2012)*. VDE-Verlag, 2012, pages 343–348.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6309532

[21] Website: http://iridia.ulb.ac.be/supp/IridiaSupp2017-004/index.html