UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

PRESENTATION PLAN
INDIVIDUAL DISSERTATION

May 2021

INDIVIDUAL DISSERTATION

MONITORING SYSTEM FOR INDUSTRIAL IOT APPLICATIONS

Andreas Komis

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2021

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

Monitoring System for Industrial IoT Applications

Andreas Komis

Supervisor

Marios D. Dikaiakos

The individual dissertation has been submitted to partially fulfill the requirements for acquiring a bachelor's degree in Computer Science by the University of Cyprus.

May 2021

Thanks

I would like to thank my family and friends who have supported me throughout my academic journey for the past 4 years. I would also like to thank the people of the Centre for Entrepreneurship for providing all the tools necessary for the completion of this project through Makerspace.

Abstract

A monitoring system that collects data from devices such as the Dobot Magician, Sliding Rail and JeVois Camera that are used in an industrial setting using Prometheus for monitoring device metrics. By using a visualization option compatible with Prometheus such as Grafana one can visualize such metrics and provide better insight. The goal of this system is to be an effective, modular and extensible solution at monitoring such devices. This project aims at doing that while being efficient, without interfering with the normal operations of the devices, and be scalable in terms of the number of devices that can be monitored as well as the number of different types of devices it supports. Also provide high monitoring flexibility through a plethora of configuration options for the monitoring agent and what device attributes to be extracted and monitored, configured individually for each device.

Contents

Chapter 1

Introduction

1.1 Motivation

One of the most timely and exciting advancements in technology is Internet of Things, something that can be observed by the ever-increasing amount of connected IoT devices around the globe. Internet of Things is a technology paradigm where a plethora of devices, digital or not, have the capability to connect to the internet, exchange information and thus be able to interact with each other [2]. This interconnectivity that the IoT concept provides, due to its versatility of devices and the possible combinations creates many applications for it. One of the most prominent applications is the industrial area where assembly (or other) lines consisting of multiple devices that cooperate are used to accomplish an end goal. This is also referred to as Industrial Internet of Things (IIoT) and is defined by L. Aberle as *"the use of Internet of Things (IoT) technologies in manufacturing"* [3] and more precisely by Hugh Boyes as "*A system comprising networked smart objects, cyber-physical assets, associated generic information technologies and optional cloud or edge computing platforms, which enable real-time, intelligent, and autonomous access, collection, analysis, communications, and exchange of process, product and/or service information, within the industrial environment, so as to optimise overall production value*" [1]. Usually in an industrial application the setup is of high scale with multiple devices for multiple purposes and dependencies throughout the work line. When dealing with such workflows it becomes hard to manage them efficiently manually and automation of this process becomes a necessity. The base for proper management is monitoring and especially in an industrial setting where there are various and complex applications. Apart from having a monitoring system take care of observing the devices and their state in an automatic fashion, proper monitoring of the devices can have many benefits that helps users manage, maintain, and optimize their workflows.

1.2 Benefits of monitoring

Monitoring is essentially about extracting, storing and observing data from a device, most commonly sensors, on a regular basis. This data is stored in specific ways for a computing device to then analyze such data and create useful information in different ways. Some common scenarios are visualizing the monitoring data for a human to observe in real time and get information about the state of certain devices. Another use case to handling this data is to define thresholds to values of the monitored device that will trigger alarms/notifications in order to either act upon a situation manually or program it further to automatically adjust in a certain way through software in order to deal with a potential issue. This is exceptionally useful when there are many nodes in a network that have certain dependencies such as a network of IoT devices. Last but not least, statistical analysis to such data can yield very useful information for the end user that can help optimize a system as well as help with decision making in regard to managing such environments.

1.3 Technical challenges

To properly monitor the workflow, the system must be able to collect data from multiple devices through a standard routine (based on intervals) and allocate its resources fairly throughout the monitoring period. This procedure can be challenging as different devices use different interfaces to connect and communicate with the host something that increases the complexity of the setups. Moreover, aggressive fetching can result in a system that misses useful data throughout the routine and/or affects the normal operations of the devices by throttling them through excessive fetching rates. Apart from this, monitoring adds overhead to both the devices and the host machine running the monitoring agent side effects e.g. increase in device temperature or interrupting the normal operations of the devices.

1.4 Project's goals

The goal of this system is to be a scalable, modular and extensible solution at monitoring devices such as the Dobot Magician and the JeVois camera. This project aims at doing that while being efficient, without interfering with the normal operations of the devices, and be scalable in terms of the number of devices that can be monitored as well as the number of different types of devices it supports. Also provide high monitoring flexibility through a plethora of configuration options for the monitoring agent and what device attributes to be extracted and monitored, configured individually for each device.

1.5 Abstract Architecture

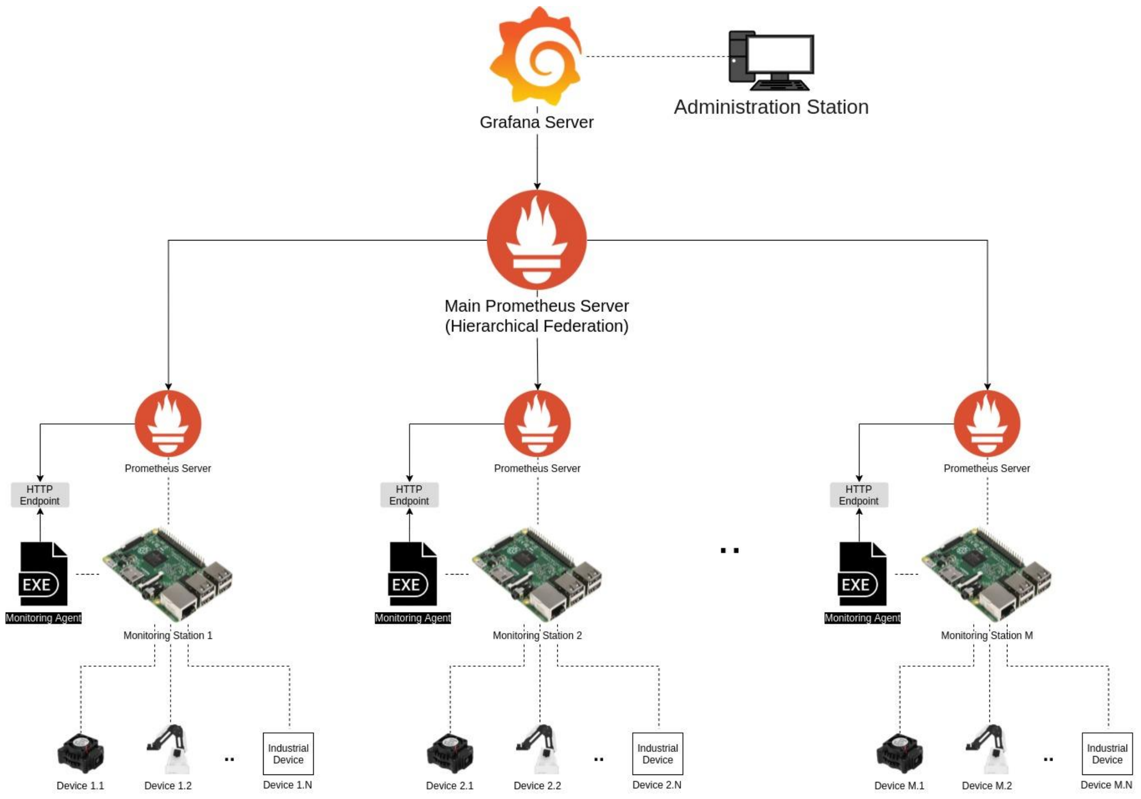


Image 1.1: Abstract architecture

The system comprise devices (that are to be monitored), host computer(s) on the edge (in close proximity to the devices) that are connected to said devices and run a monitoring agent (will be referred to as monitoring stations) and a host computer acting as the administration station which collects the metrics from each individual station using a Prometheus Hierarchical Federation (see 4.3) as a data source and visualizes them. For the monitoring of the metrics each monitoring station runs a local Prometheus server. The system can scale vertically (monitoring station level) as the agent can connect to and monitor a variable number of different devices, a number constrained by the monitoring station's available ports and resources. In addition, for larger and more complex setups one can scale the system horizontally (main monitoring station level) by adding multiple monitoring stations. For the configuration of these stations one can tweak their Prometheus and agent configuration files.

Chapter 2

Monitored Devices: Dobot Magician & JeVois Camera

2.1 Dobot Magician

2.1.1 Introduction to Dobot Magician

Dobot magician is best described in a sentence by the maker as "*a multifunctional desktop robotic arm for practical training education*" [4]. The arm can be extended by changing its end effectors. Some of them are the suction cup, the gripper, pen holder and more. With this variety of compatible equipment, the Dobot Magician can be used for 3D printing, laser engraving, writing/drawing and more. In addition to those the robot is also compatible with external equipment such as the conveyor belt kit which includes a conveyor belt, a color sensor and an infrared sensor that helps setup small scale environments that replicate an industrial production line. There are numerous kits and extensions to the Dobot Magician such as the Sliding Rail Kit that enables the robot to slide instead of being stationary. It is worth noting that the Dobot Magician has been awarded[4] the 2018 Innovation Award by CES as well as the iF DESIGN AWARD 2018, Red Dot Design Award 2018 and 2017 Red Star Design Award.

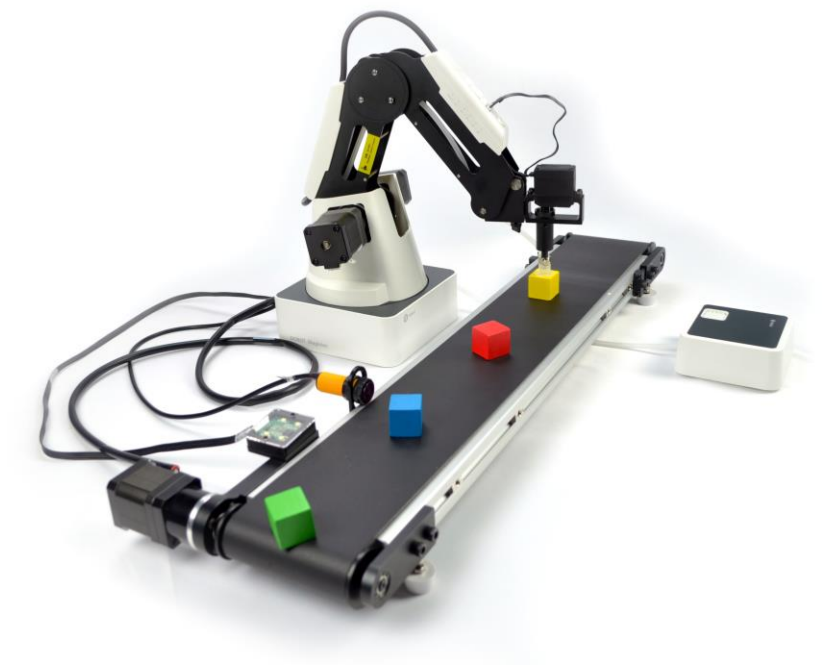


Image 2.1: Dobot Magician Robotic Arm & Conveyor Belt Kit [5]

2.1.2 Dobot Magician API

In order to interact with the arm and connected peripherals through software the Dobot Magician API[6] can be utilized. The main library that enables such communication with the device is a dynamic link library called DobotDll.dll and libDobotDll.so in Windows and Linux respectively. This library utilizes the Dobot Communication Protocol[7] in order to communicate properly with the robot. For convenience and flexibility there are many encapsulations of it in many languages such as Python, Java, C# and more that provide a more abstract way (depending on the language architecture) to interact with the robot. All the API encapsulations and their respective demos can be found in the official Dobot Magician page at "Dobot Demo v2.2"[8] under the "Development Protocol" section. The API provides the programmer with the ability to give input to the robot and control its actions as well as retrieve information about the current state of the robot and its peripherals which is essentially what enables the monitoring of this device. Such getter functions/commands can be issued to retrieve information about the positions, speed and acceleration of the joints, alarm states, and other information about the robot. For the rest of the paper when referring to the Dobot Magician API we mean the Python encapsulation of the Dobot API, DobotDllType.py, which is used by the monitoring agent for fetching all the necessary data.

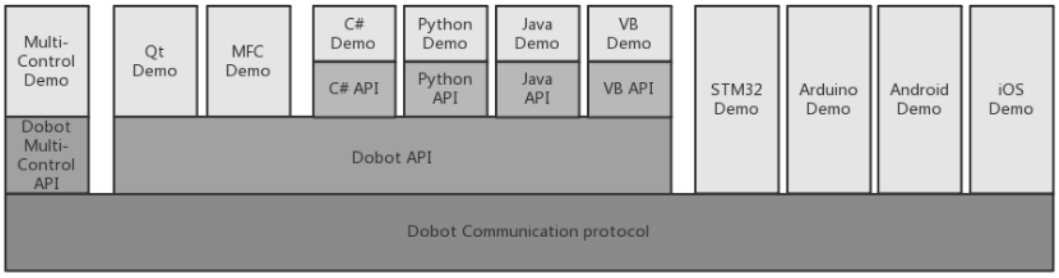


Image 2.2: Dobot Magician API Architecture Overview [8]

2.1.3 API Analysis

For better understanding what the API provides, that can be useful for monitoring, the Dobot API source file can be analyzed to extract the following information which played a crucial role in the decision making regarding the monitoring options. The API includes a total of 79 "getter" functions for various attributes of the device that can potentially be useful for monitoring. The following analysis has been done on those functions to better understand and determine which API calls can be utilized by excluding certain functions based on the following criteria.

Criteria	Number of functions
Not regarding the Dobot Magician (functions regarding the Dobot Magician Lite – another Dobot model)	15
Not included/documented in the latest Dobot Magician API Description (v1.2.3)	17
IO related and/or missing argument description and/or not useful for monitoring	11

Table 2.1: Number of excluded functions in the Dobot API based on criteria

Based on the above table we can conclude that 44 of the getter functions are not suitable for use with the monitoring agent. This implies that the remaining 35 getter functions can be utilized in the monitoring agent. From the included functions we can retrieve 83 useful info/data from the Dobot Magician as most api calls return more than one attribute. Throughout this analysis it is ensured that the monitoring agent is not bloated with useless or uninteresting calls while still providing a plethora of solid options. For more details regarding which attributes can be monitored and what API calls are responsible and used in the monitoring agent, refer to section 2.1.3.

2.1.4 API Fork

Throughout the analysis of the Dobot API, some minor issues arose with fetching certain useful attributes, due to errors in the API. Fixing those errors to not sacrifice any wanted data led to a greater understanding of how the Dobot API works and resulted to more changes that make the Dobot API more flexible and more convenient to use. No functions are changed from the original Dobot API as to not break any existing implementations utilizing the official API, as all changes to functions are done through wrappers (new functions that utilize the existing functions with additions). For using the improved functions provided by the fork one should create a "runtime" directory in the working directory (where the python script utilizing it will run) with all the files provided in the Dobot Demo.

Fixes

- Fixed GetPoseL(api) function, which returns the position of the sliding rail (if there is one connected to the robot), by importing the math library which is required for the needs of the function, however not included by default.

Improvements

- Created function loadX() to replace load(). It has been observed that the basic information for communicating with the Dobot is set by ConnectDobot() and are stored in this file for future api calls. By calling the default load() this file is overridden each time which makes parallel connections impossible. loadX() implements loading individual dll/so (DobotDll.dll instances) for each connected device allowing parallel connection with multiple Dobots. This function is not meant to be called explicitly. Connection can be made through ConnectDobotX(port) which uses it properly.
- Created function ConnectDobotX(port) to replace "api = load(); state = ConnectDobot(api, port, baudrate)" for connecting to a Dobot Magician device. The main improvement this change provides is that through its implementation,

by utilizing the loadX(), it allows parallel connections to Dobot Magicians and also removes the need to issue separately. When using the default API this model is not feasible and multiple Dobot Magicians can be connected concurrently with a switching overhead of approximately 0.3 seconds per switch. Apart from the performance benefits this function provides, it is also a more readable and convenient option for connecting a Dobot Magician device as all the standardized procedures are included either in the function or through default arguments.

- Created function GetAlarmsStateX(api) which is an alternative to GetAlarmsState(api, maxLen) that uses a hardcoded dictionary of bit addresses and alarm descriptions is used for decoding the byte array returned by the default function and instead return the active alarms in human readable form (alarm name and description). The decoding of the alarms byte array is achieved by traversing the array by alarm index based on a hardcoded dictionary called alarms{} with the key being the bit index and the corresponding value the alarm description as described in the Dobot ALARM document [21].

Additions

- Created function GetActiveDobots() that returns the amount of currently connected Dobot Magician.
- Created function DisconnectAll() to disconnect from all connected Dobot Magician devices and clean up any runtime files.

  Both additions were due to the major ConnectDobotX(port) function and their purpose is to accommodate it and enrich the flexibility it provides.

2.1.5 Monitoring options

After the API analysis described in section 2.1.2.2 the following 82 attributes of a Dobot Magician device are supported for monitoring.

Description	**API Call**
Device's serial number	GetDeviceSN(api)
Device's name/alias	GetDeviceName(api)
Device's version (major.minor.0.revision)	GetDeviceVersion(api)
Device's clock/time	GetDeviceTime(api)

Description	**API Call**
Current index in command queue	GetQueuedCmdCurrentIndex(api)
Real-time cartesian coordinate of device's X axis	GetPose(api)
Real-time cartesian coordinate of device's Y axis	GetPose(api)
Real-time cartesian coordinate of device's Z axis	GetPose(api)
Real-time cartesian coordinate of device's R axis	GetPose(api)
Base joint angle	GetPose(api)
Rear arm joint angle	GetPose(api)
Forearm joint angle	GetPose(api)
End effector joint angle	GetPose(api)
Device's active alarms	GetAlarmsState(api)
Home position for X axis	GetHOMEParams(api)
Home position for Y axis	GetHOMEParams(api)
Home position for Z axis	GetHOMEParams(api)
Home position for R axis	GetHOMEParams(api)
X-axis offset of end effector	GetEndEffectorParams(api)
Y-axis offset of end effector	GetEndEffectorParams(api)
Z-axis offset of end effector	GetEndEffectorParams(api)
Status (enabled/disabled) of laser	GetEndEffectorLaser(api)
Status (enabled/disabled) of suction cup	GetEndEffectorSuctionCup(api)
Status (enabled/disabled) of gripper	GetEndEffectorGripper(api)
Velocity (°/s) of base joint in jogging mode	GetJOGJointParams(api)
Velocity (°/s) of rear arm joint in jogging mode	GetJOGJointParams(api)
Velocity (°/s) of forearm joint in jogging mode	GetJOGJointParams(api)
Velocity (°/s) of end effector joint in jogging mode	GetJOGJointParams(api)
Acceleration (°/s^2) of base joint in jogging mode	GetJOGJointParams(api)
Acceleration (°/s^2) of rear arm joint in jogging mode	GetJOGJointParams(api)
Acceleration (°/s^2) of forearm joint in jogging mode	GetJOGJointParams(api)
Acceleration (°/s^2) of end effector joint in jogging mode	GetJOGJointParams(api)

Description	API Call
Velocity (mm/s) of device's X axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Velocity (mm/s) of device's Y axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Velocity (mm/s) of device's Z axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Velocity (mm/s) of device's R axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Acceleration (mm/s^2) of device's X axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Acceleration (mm/s^2) of device's Y axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Acceleration (mm/s^2) of device's Z axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Acceleration (mm/s^2) of device's R axis (cartesian coordinate) in jogging mode	GetJOGCoordinateParams(api)
Velocity ratio of all axis (joint and cartesian coordinate system) in jogging mode	GetJOGCommonParams(api)
Acceleration ratio of all axis (joint and cartesian coordinate system) in jogging mode	GetJOGCommonParams(api)
Velocity (°/s) of base joint in point to point mode	GetPTPJointParams(api)
Velocity (°/s) of rear arm joint in point to point mode	GetPTPJointParams(api)
Velocity (°/s) of forearm joint in point to point mode	GetPTPJointParams(api)
Velocity (°/s) of end effector joint in point to point mode	GetPTPJointParams(api)
Acceleration (°/s^2) of base joint in point to point mode	GetPTPJointParams(api)
Acceleration (°/s^2) of rear arm joint in point to point mode	GetPTPJointParams(api)
Acceleration (°/s^2) of forearm joint in point to point mode	GetPTPJointParams(api)
Acceleration (°/s^2) of end effector joint in point to point mode	GetPTPJointParams(api)
Velocity (mm/s) of device's X, Y, Z axis (cartesian coordinate) in point to point mode	GetPTPCoordinateParams(api)
Velocity (mm/s) of device's R axis (cartesian coordinate) in point to point mode	GetPTPCoordinateParams(api)

Description	**API Call**
Acceleration (mm/s^2) of device's X, Y, Z axis (cartesian coordinate) in point to point mode	GetPTPCoordinateParams(api)
Acceleration (mm/s^2) of device's R axis (cartesian coordinate) in point to point mode	GetPTPCoordinateParams(api)
Velocity ratio of all axis (joint and cartesian coordinate system) in point to point mode	GetPTPCommonParams(api)
Acceleration ratio of all axis (joint and cartesian coordinate system) in point to point mode	GetPTPCommonParams(api)
Lifting height in jump mode	GetPTPJumpParams(api)
Max lifting height in jump mode	GetPTPJumpParams(api)
Velocity (mm/s) in cp mode	GetCPParams(api)
Acceleration (mm/s^2) in cp mode	GetCPParams(api)
Velocity (mm/s) of X, Y, Z axis in arc mode	GetARCParams(api)
Velocity (mm/s) of R axis in arc mode	GetARCParams(api)
Acceleration (mm/s^2) of X, Y, Z axis in arc mode	GetARCParams(api)
Acceleration (mm/s^2) of R axis in arc mode	GetARCParams(api)
Rear arm angle sensor static error	GetAngleSensorStaticError(api)
Forearm angle sensor static error	GetAngleSensorStaticError(api)
Rear arm angle sensor linearization parameter	GetAngleSensorCoef(api)
Forearm angle sensor linearization parameter	GetAngleSensorCoef(api)
Sliding rail's status (enabled/disabled)	GetDeviceWithL(api)
Sliding rail's real-time pose in mm	GetPoseL(api)
Velocity (mm/s) of sliding rail in jogging mode	GetJOGLParams(api)
Acceleration (mm/s^2) of sliding rail in jogging mode	GetJOGLParams(api)
Velocity (mm/s) of sliding rail in point to point mode	GetPTPLParams(api)
Acceleration (mm/s^2) of sliding rail in point to point mode	GetPTPLParams(api)
Wifi module status (enabled/disabled)	GetWIFIConfigMode(api)
Wifi connection status (connected/not connected)	GetWIFIConnectStatus(api)
Configured Wifi SSID	GetWIFISSID(api)
Configured Wifi Password	GetWIFIPassword(api)

Description	API Call
Device's IP address	GetWIFIIPAddress(api)
Device's configured subnet mask	GetWIFINetmask(api)
Device's configured default Gateway	GetWIFIGateway(api)
Device's configured DNS	GetWIFIDNS(api)

Table 2.1: Dobot Magician Supported Monitoring Options

2.1.6 Connectivity

The Dobot Magician supports connection to a host computer through USB to serial port (with the corresponding included cable), WiFi and Bluetooth[7]. In order to achieve a wireless connection to the robot, either through Bluetooth or WiFi, the corresponding kit needs to be installed to the device by connecting the corresponding adapter to the UART interface on the base and restarting the device. The Wireless-1 adapter corresponds to Bluetooth kit and the Wireless-2 adapter to the WiFi kit. The green light on the adapters indicate that the device is currently connected, and communication can be made. The Bluetooth connectivity is not related to this project as it is only used to communicate with the robot through mobile applications. For a Dobot Magician to be connected through WiFi to a host computer both the robot and the host computer must be connected to the same local network (WLAN). A convenient approach to connecting a Dobot Magician through WiFi is to first connect the Dobot through USB to serial port in the host computer and then use the Magician Studio[9] (Windows only) provided by Dobot as an interface to setup the WiFi settings (e.g. the IP address of the robot) and test the connection through a GUI environment before proceeding with any monitoring. The other option is to configure the WiFi settings by connecting through USB and making the appropriate API calls (for configuring the IP, default gateway, DNS etc).

Image 2.2: Dobot Magician WiFi Kit (left) and

Dobot Magician successfully connected to the network indication (right) [10]

2.2 JeVois Camera

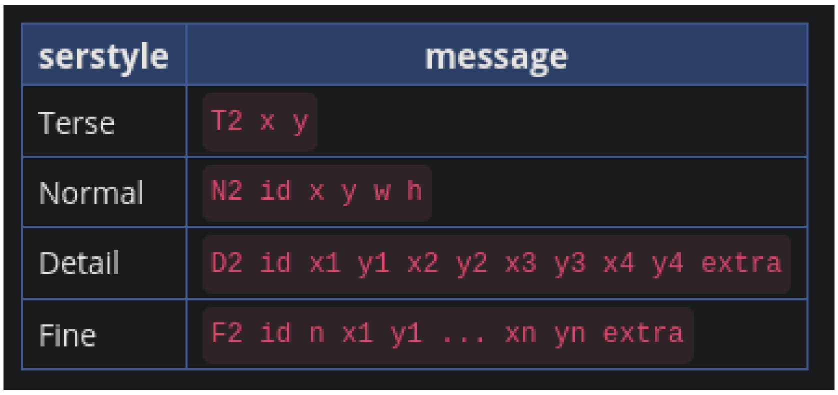
2.2.1 Introduction to JeVois Camera

The JeVois camera is a smart machine vision camera with many capabilities. It is composed by a 1.3 megapixels video sensor, an ARM Cortex A7 quad-core CPU, 256 MB RAM and usb to serial port. It is very lightweight and has a small size of 28 cm3 which makes it extremely portable and versatile. The camera has an SD card input where through configuration files the desired module can be loaded and executed by the camera. There are many different modules including locating and identifying faces and objects, classifying different object types, detecting and decoding QR codes and more. The camera can operate to different resolutions for different frame rates up to 120 frames per second depending on the use case and its requirements. For the retrieval and processing of information from the JeVois camera there is the ability to extract the results of the camera through structured messages, something that is the main use of the camera for the purpose this project.[11]

Image 2.3: JeVois Smart Machine Vision Camera[11]

2.2.2 Standardized serial messages

For machine to machine interaction JeVois doesn't provide some sort of API, however its modules have the ability to produce standardized set of serial output messages. The current focus of this mechanism is about sending identity and location information in either 1, 2 or 3 (experimental) dimensions. By altering the serstyle and serprec parameters in the jevois::StdModule one can define different styles (formats) of the messages. Because of this customization of formats the monitoring agent needs to first identify the type of message (object detection/recognition and/or location), if it is a location message the dimensions of the location (1D, 2D or 3D) and then the serstyle (Terse, Normal, Detail, Fine) in order to handle the message (with the possibility of rejecting it), potentially parse the message and further monitor the data properly.[12] Standardized location serial messages follow the following format "[S][D] [data..]", where [S] is a capital character representing the serstyle T, N, D, F for Terse, Normal, Detail and Fine respectively and [D] is a single-digit number (character) representing the number of dimensions 1, 2, 3 for 1D, 2D, 3D respectively. For different serstyle examples see the image 2.4. The available serstyle and dimensions of the message are constrained by the active module implementation. The user can change the serstyle of the output, among other settings, by modifying the initscript.cfg file on the JeVois camera. The rate in which standardized messages are produced is one per frame which allows for real-time calculations. If there is no identified object then the standardized message is empty.



serstyle	message
Terse	`T2 x y`
Normal	`N2 id x y w h`
Detail	`D2 id x1 y1 x2 y2 x3 y3 x4 y4 extra`
Fine	`F2 id n x1 y1 ... xn yn extra`

Image 2.4: Two-dimensional (2D) location messages[12]

2.2.3 Monitoring options

Following the previous section, we can conclude that the only information we can extract from the camera are the results in regards of location and identity of the identified object. standardized positions x, x/y and x/y/z can be monitored for 1D, 2D and 3D messages, respectively. With the location data we can estimate the object's size as well. The size is calculated differently depending on the number of dimensions of the message. If the location message is one-dimensional then the size is provided by the message[12]. For more dimensions, the size is calculated by multiplying the width/height and width/height/depth for 2D and 3D location messages respectively.

2.2.4 Connectivity

The JeVois Camera can transmit data either through a 4-pin JST-SH 1.0mm connector (micro-serial connector) mostly used for connecting it to a microcontroller or a mini-USB

2.0 type B 5-pin connector. For the needs of the project, we will be utilizing the latter. As the JeVois camera can be considered a standalone computer since it has an embedded CPU and RAM, there is need for both power and data transmission. Depending on the setup one can provide both through a USB 3.0 port or 2 USB 2.0 ports of a host computer. For the scenario where there are limited USB 2.0 ports, one can provide power through an adaptor on a socket or a power bank.

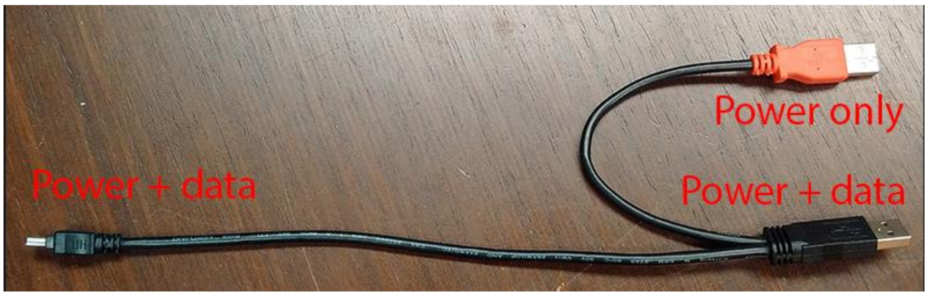


Image 2.5a: USB 2.0 Y-type cable for connecting to host[13]

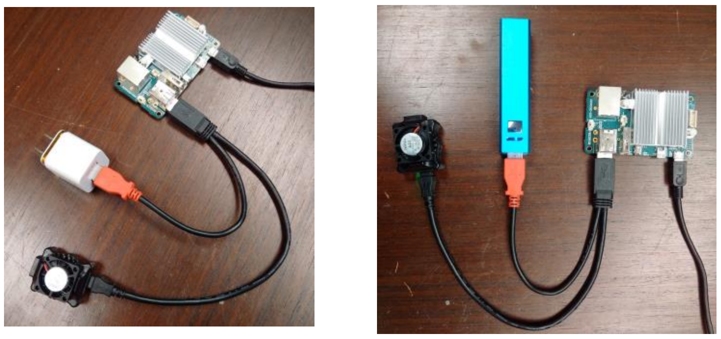


Image 2.5b: Connection to host and power through socket (left) and through powerbank (right)[13]

Chapter 3

Middleware: Monitoring Agent

3.1 Responsibilities

The monitoring agent is a piece of software written in Python mainly responsible of extracting the data from the monitored devices in specified intervals without interfering with the normal operations of the devices. It primarily acts as the middleware between the devices and the main monitoring system (see Chapter 4). Apart from this main goal, the monitoring agent aims at providing flexibility and customizability for which monitored data to extract for each individual device as to be usable in many different scenarios. The agent also produces related output to standard output and standard error for real-time updates on the process as well as logging.

3.2 Dependencies

The implementation of the agent tries to minimize the number of dependencies. The monitoring agent uses the sys, time, argparse, webbrowser, configparser and threading modules from the Python Standard Library and more specifically works with Python v3.9 or greater. Another, non-standard, dependency of the monitoring agent is the prometheus_client library. For each device type and thus device module (in device_modules.py), different dependencies are needed. The Jevois camera device module (class Jevois) depends on the pyserial module and for the Dobot Magician device module (class Dobot), the Dobot python module the Dobot Robot Driver (which one convenient way to be acquired is by installing the Magician Studio), the Dobot API and the DobotTypeDllX.py (python encapsulation fork) with the corresponding library objects.

3.3 Usage

When using the default configuration file location, make sure that "devices.conf" file is properly setup and in the same directory as the executable. For configuring the agent one can pass the following command line arguments. All arguments are optional. For arguments that a default value is specified, a value is needed when using them.

Usage: $ agent.py [-d DEVICES] [-n NAME] [-p PROMPORT] [-k] [-v] [-m] [-h]

Notation		**Argument Description**	**Default**
Short	**Long**		
-d	--devices	Specify discovery/configuration file absolute path	devices.conf
-n	--name	Specify symbolic agent/station name	Agent0
-p	--promport	Specify port number for the Prometheus endpoint	8000
-k	--killswitch	Exit agent if error occurs in validation/connection phase	-
-v	--verbose	Print actions with details in standard output	-
-c	--color	Print color rich messages to terminal (ANSI escape colors)	-
-m	--more	Open README.md with configuration and implementation details	-
-h	--help	Show help message with command line arguments and exit	-

Table 3.1: Agent command line arguments

3.4 Device discovery & configuration

For device discovery and configuration of which data to be monitored for each device the agent uses a configuration file which default name is "devices.conf". This configuration file follows a structure similar to Microsoft Windows INI[14] files and is parsed in the agent by using configparser[15]. This format has been chosen as it provides a simple and understandable configuration interface that fulfills all the configuration needs of the monitoring agent, as there are no needs for nested configuration options in which case a format like JSON would be more appropriate. The configuration file has no mandatory fields as there are default fallback values for each option, giving the ability for the user to completely exclude options from the configuration file for better readability. The configuration file is divided in sections, each section representing a device. Regarding the device entries, for enabling data to be monitored one can use \`on\`, \`1\`, \`yes\` or \`true\` and in order to not monitor certain data use \`off\`, \`0\`, \`no\`, \`false\` based on personal preference. By removing an entry completely, the value for the entry will be resolved to the default. All keys are case-insensitive, but all section names must be identical to the class name of the device module. All keys in the configuration file are case-insensitive as they are always stored in lower case, however all section titles must be the same as their respective device module class. For all device attributes the Timeout can be used to set a custom timeout in milliseconds in between fetch calls, which defaults to 0. All configuration is parsed and validated based on the above information, before the start of the routine, and warns the user for any invalid entries, fields and values. It is worth noting that comments are supported and can be used by starting the line with the character '#'. For a more practical view, see Appendix E.

3.4.1 Dobot Magician

For connecting to a Dobot Magician device there should be a [Dobot:<port>] where <port> can be either a serial communication port (e.g. [Dobot:COM3]) or an IP address (e.g. [Dobot:192.168.0.3]). The default configuration options are such as to provide a minimal monitoring setting with only basic, necessary and non-situational attributes enabled. Table 3.2 contains a list of all the configuration options for Dobot Magician devices and their details.

Config Name	Description	Default
DeviceSN	Device's serial number	on
DeviceName	Device's name/alias	on
DeviceVersion	Device's verion (major.minor.0.revision)	on
DeviceTime	Device's clock/time	off
QueueIndex	Current index in command queue	off
PoseX	Real-time cartesian coordinate of device's X axis	on
PoseY	Real-time cartesian coordinate of device's Y axis	on
PoseZ	Real-time cartesian coordinate of device's Z axis	on
PoseR	Real-time cartesian coordinate of device's R axis	on
AngleBase	Base joint angle	on
AngleRearArm	Rear arm joint angle	on
AngleForearm	Forearm joint angle	on
AngleEndEffector	End effector joint angle	on
AlarmsState	Device's active alarms	on
HomeX	Home position for X axis	off
HomeY	Home position for Y axis	off
HomeZ	Home position for Z axis	off
HomeR	Home position for R axis	off
EndEffectorX	X-axis offset of end effector	off
EndEffectorY	Y-axis offset of end effector	off
EndEffectorZ	Z-axis offset of end effector	off
LaserStatus	Status (enabled/disabled) of laser	off
SuctionCupStatus	Status (enabled/disabled) of suction cup	off
GripperStatus	Status (enabled/disabled) of gripper	off
JogBaseVelocity	Velocity (°/s) of base joint in jogging mode	off
JogRearArmVelocity	Velocity (°/s) of rear arm joint in jogging mode	off
JogForearmVelocity	Velocity (°/s) of forearm joint in jogging mode	off
JogEndEffectorVelocity	Velocity (°/s) of end effector joint in jogging mode	off
JogBaseAcceleration	Acceleration (°/s^2) of base joint in jogging mode	off

Config Name	Description	Default
JogRearArmAcceleration	Acceleration (°/s^2) of rear arm joint in jogging mode	off
JogForearmAcceleration	Acceleration (°/s^2) of forearm joint in jogging mode	off
JogEndEffectorAcceleration	Acceleration (°/s^2) of end effector joint in jogging mode	off
JogAxisXVelocity	Velocity (mm/s) of device's X axis (cartesian coordinate) in jogging mode	off
JogAxisYVelocity	Velocity (mm/s) of device's Y axis (cartesian coordinate) in jogging mode	off
JogAxisZVelocity	Velocity (mm/s) of device's Z axis (cartesian coordinate) in jogging mode	off
JogAxisRVelocity	Velocity (mm/s) of device's R axis (cartesian coordinate) in jogging mode	off
JogAxisXAcceleration	Acceleration (mm/s^2) of device's X axis (cartesian coordinate) in jogging mode	off
JogAxisYAcceleration	Acceleration (mm/s^2) of device's Y axis (cartesian coordinate) in jogging mode	off
JogAxisZAcceleration	Acceleration (mm/s^2) of device's Z axis (cartesian coordinate) in jogging mode	off
JogAxisRAcceleration	Acceleration (mm/s^2) of device's R axis (cartesian coordinate) in jogging mode	off
JogVelocityRatio	Velocity ratio of all axis (joint and cartesian coordinate system) in jogging mode	off
JogAccelerationRatio	Acceleration ratio of all axis (joint and cartesian coordinate system) in jogging mode	off
PtpBaseVelocity	Velocity (°/s) of base joint in point to point mode	off
PtpRearArmVelocity	Velocity (°/s) of rear arm joint in point to point mode	off
PtpForearmVelocity	Velocity (°/s) of forearm joint in point to point mode	off
PtpEndEffectorVelocity	Velocity (°/s) of end effector joint in point to point mode	off
PtpBaseAcceleration	Acceleration (°/s^2) of base joint in point to point mode	off
PtpRearArmAcceleration	Acceleration (°/s^2) of rear arm joint in point to point mode	off
PtpForearmAcceleration	Acceleration (°/s^2) of forearm joint in point to point mode	off
PtpEndEffectorAcceleration	Acceleration (°/s^2) of end effector joint in point to point mode	off

Config Name	Description	Default
PtpAxisXYZVelocity	Velocity (mm/s) of device's X, Y, Z axis (cartesian coordinate) in point to point mode	off
PtpAxisRVelocity	Velocity (mm/s) of device's R axis (cartesian coordinate) in point to point mode	off
PtpAxisXYZAcceleration	Acceleration (mm/s^2) of device's X, Y, Z axis (cartesian coordinate) in point to point mode	off
PtpAxisRAcceleration	Acceleration (mm/s^2) of device's R axis (cartesian coordinate) in point to point mode	off
PtpVelocityRatio	Velocity ratio of all axis (joint and cartesian coordinate system) in point to point mode	off
PtpAccelerationRatio	Acceleration ratio of all axis (joint and cartesian coordinate system) in point to point mode	off
LiftingHeight	Lifting height in jump mode	off
HeighLimit	Max lifting height in jump mode	off
CpVelocity	Velocity (mm/s) in cp mode	off
CpAcceleration	Acceleration (mm/s^2) in cp mode	off
ArcXYZVelocity	Velocity (mm/s) of X, Y, Z axis in arc mode	off
ArcRVelocity	Velocity (mm/s) of R axis in arc mode	off
ArcXYZAcceleration	Acceleration (mm/s^2) of X, Y, Z axis in arc mode	off
ArcRAcceleration	Acceleration (mm/s^2) of R axis in arc mode	off
AngleStaticErrRear	Rear arm angle sensor static error	off
AngleStaticErrFront	Forearm angle sensor static error	off
AngleCoefRear	Rear arm angle sensor linearization parameter	off
AngleCoefFront	Forearm angle sensor linearization parameter	off
SlidingRailStatus	Sliding rail's status (enabled/disabled)	off
SlidingRailPose	Sliding rail's real-time pose in mm	off
SlidingRailJogVelocity	Velocity (mm/s) of sliding rail in jogging mode	off
SlidingRailJogAcceleration	Acceleration (mm/s^2) of sliding rail in jogging mode	off
SlidingRailPtpVelocity	Velocity (mm/s) of sliding rail in point to point mode	off
SlidingRailPtpAcceleration	Acceleration (mm/s^2) of sliding rail in point to point mode	off

Config Name	Description	Default
WifiModuleStatus	Wifi module status (enabled/disabled)	off
WifiConnectionStatus	Wifi connection status (connected/not connected)	off
WifiSSID	Configured Wifi SSID	off
WifiPassword	Configured Wifi Password	off
WifiIPAddress	Device's IP address	off
WifiNetmask	Device's configured subnet mask	off
WifiGateway	Device's configured default Gateway	off
WifiDNS	Device's configured DNS	off

Table 3.2: Dobot Magician configuration settings for monitoring options

3.4.2 JeVois Camera

For connecting to a JeVois Camera device there should be a [Jevois:<port>] where <port> can be strictly a serial communication port (e.g. [Jevois:COM3]) since the device doesn't natively support connecting through WiFi or wirelessly in general. Under the section one can configure which data to be monitored by using the same notations as described in 3.2.1. The location data includes the standardized positions, x for 1D messages, x/y for 2D messages and x/y/z for 3D messages thus the number of dimensions of the location dictate the monitored metrics. In order to properly monitor what type of object has been identified the user must include a list of object names divided by space under the device section in an entry called "objects", that must be the names of the images' (file) names included in the target set existing in the "Saved" directory on the JeVois' SD card e.g. "objects = cube pen paper". For better readability whilst monitoring it is recommended that those filenames do not include a file extension and the image names concisely describe the object. Table 3.3 has a list of all the configuration options for the JeVois Camera and their details.

Config Name	Description	Default
ObjectIdentified	Identified object's name	on
ObjectLocation	Identified object's location	on
ObjectSize	Identified object's size	off

Table 3.3: JeVois Camera configuration settings for monitoring options

Currently the monitoring agent, due to to the implementation of the Jevois module, only supports location messages with the Normal serstyle, since other serstyles like Terse provide insufficient data for the monitoring options provided and the Detail and Fine serstyles provide excess data that would further complicate the Jevois module's implementation for no great benefit, something that would go against the monitoring agent's philosophy. However, there is support for all available dimensions (N1/N2/N3 types of messages) to provide a decent amount of flexibility.

3.4.3 Other Device

For custom device classes in the device_modules.py e.g. DeviceType, a [DeviceType:<port>] entry must exist in the configuration for the monitoring agent to automatically discover it and use the appropriate module for connecting, fetching and disconnecting from said device (see 3.6).

3.5 Procedures

Firstly, the monitoring agent opens, parses and validates the configuration file through the __validateConfig() function which uses the respective options directory for each device to verify that the configuration file is valid and warn the user about validation issues before the start of the routine. Validation is being done to check that the fields of a device section are valid (such options are supported by the module), that the value of that field is of the correct type, that the respective device module exists and implement the Device class properly and that at least 1 monitored option is enabled. The validSections list is created by __validateConfig() including only the device (config) sections with no errors. Based on this list the agent uses the appropriate device module (from device_modules.py) to connect to a device through the specified port and adds it to the devices[] list which holds all connected device objects. Finally, the agent spawns a thread for each device object and fetches the enabled attributes with the specified per device timeout in between its routine.

3.6 Extensibility

The agent currently supports Dobot Magician and JeVois Camera devices. For extending the agent's capabilities to support a different type of device one can create a device class (device module) and place it in the device_modules.py. This class needs to be a child of the Device class (found in the same file) and implement all its attributes and methods. The name of the class is determining the name that the agent will use to discover a device through agent.conf, connect to it, fetch (and inform prometheus) its attributes and finally disconnect from the device. The only member that need to be implemented is the options{} dictionary and the connect(), fetch() and disconnect() methods.

- options{}: A dictionary that includes all the valid fields/options a device can have in the configuration file (monitored attribute fields) as keys and their default value (also used to validate the type of the value in the configuration file) as values.
- connect(): Responsible for connecting to the device, initialize any Prometheus metrics and other necessary device information that is vital for the use of the other methods. If the connection attempt is unsuccessful it should raise an exception.
- fetch(): Used to extract all enabled monitoring attributes for said device and update the Prometheus metrics accordingly. If the fetch attempt is unsuccessful it should raise an exception.
- disconnect(): Responsible for disconnecting the device, close any open ports/streams and remove any runtime temporary files regarding the device.

All other necessary modules needed for implementing the above functions (e.g. DobotDllTypeX.py for the Dobot device module) and any runtime files should be included in the "runtime" directory and imported properly in device_modules.py. For a practical example one can review the source code in device_modules.py. Also note that all Prometheus objects for recording metrics should be static members of the class in order to allow for label use. For a more practical view see Appendix C.

3.7 Testing

For both functional and performance tests, a small manageable testing utility has been developed in parallel with the agent’s development named test.py (see appendix D) which includes a number of functions respective to different functional (f) and performance (p) tests for different device modules. Each functional test represents a function of a device module. Each test returns true in successful completion and false otherwise. Performance tests produce results/statistics to standard output that can be further analyzed. The naming convention for better organization and use of the test functions is as follows: typeOfTest_moduleName_description

e.g. For a performance test regarding the Jevois module p_Jevois_Description()

3.8 Performance analysis

A series of performance tests to benchmark the agent and device modules were done. Memory footprint and fetch times were the two most important metrics that were taken into consideration. The performance tests were run on a x64 Windows (OS build 19041.985) machine. Both Dobot and Jevois related software (dobot driver, dobot api, jevois image) were the latest based on the date of the creation of this paper. The memory footprint of the agent prior to connecting to any devices (and thus creating any device objects) was 25MB and after connecting 2 Dobot Magicians and 1 Jevois camera with all attributes enabled it increased to 30MB. The performance/responsiveness for connecting, fetching and disconnecting from a device is determined by the respective device module implementation, the device architecture and the monitoring station's available resources.

3.8.1 Dobot

For the fetch times regarding the regarding the Dobot Magician device and the Dobot device module for all attributes the average fetch time while connected through usb was ~15ms and when connected through WiFi ~24ms, which leads to the conclusion that the general wireless overhead is around 9ms. However, for some of the WiFi related (see Tabel 3.4 and Image 3.1) and the GetDeviceTime api calls, the fetch times were lower (better) when connected wirelessly. These attributes are also the only attributes that their fetch times always exceed 500ms in both cases. Since these WiFi attributes are of type Info, which means they are only fetched once at the start of the monitoring phase this is not a bottleneck to the fetching routine. It has been observed that if the Dobot is under any kind of movement starting the monitoring with the WiFi attributes enabled will affect the normal operations of the robot (supposedly due to the high fetch times interfering with the normal command execution) so only enable these attributes if the monitoring will start prior to the normal operations. Since DeviceTime is the only non-info attribute that exceeds the average low fetch times (by a great amount), it is advised that enabling the DeviceTime option is avoided. All fetch time results were produced by calling the certain API call 30 times and calculating the min, max and average. A detailed look of the results for both wired and wireless connections can be seen in the table below.

API Call	**Wired**			**Wireless**		
	Average	**Min**	**Max**	**Average**	**Min**	**Max**
GetDeviceTime	719	505	1014	608	205	655
GetQueuedCmdCurrentIndex	15	10	20	27	18	147
GetPose	16	12	23	24	20	33
GetAlarmsStateX	17	10	20	25	19	32
GetHOMEParams	20	13	24	29	22	34
GetAutoLevelingResult	15	10	21	24	18	31
GetEndEffectorParams	15	10	22	23	18	34
GetEndEffectorLaser	13	7	20	23	17	30
GetEndEffectorSuctionCup	13	7	20	23	18	34
GetEndEffectorGripper	14	8	22	22	18	29
GetJOGJointParams	17	12	31	26	21	33
GetJOGCoordinateParams	17	9	22	28	20	33
GetJOGCommonParams	15	9	20	25	16	30
GetPTPJointParams	16	12	23	26	20	33
GetPTPCoordinateParams	17	10	24	25	20	31
GetPTPCommonParams	14	9	24	24	17	29
GetPTPJumpParams	15	9	25	24	17	31
GetCPParams	14	9	21	24	19	30
GetARCParams	16	9	21	24	18	32
GetAngleSensorStaticError	15	9	22	25	20	30
GetAngleSensorCoef	16	8	22	25	18	39
GetDeviceWithL	14	8	19	23	17	29
GetPoseL	13	8	20	22	18	31
GetJOGLParams	13	10	20	21	17	28
GetPTPLParams	13	9	20	23	18	30
GetWIFIConfigMode	14	8	20	24	18	30
GetWIFIConnectStatus	13	10	21	22	18	28

GetDeviceSN	13	9	19	24	19	40
GetDeviceName	16	11	22	26	18	34
GetDeviceVersion	16	9	20	26	20	32
GetWIFISSID	521	516	528	536	531	541
GetWIFIPassword	1676	1011	2541	1153	1046	1647
GetWIFIIPAddress	1499	1012	2016	1368	1122	1739
GetWIFINetmask	1550	1012	2031	1200	625	1727
GetWIFIGateway	1635	1013	2537	1141	1066	1630
GetWIFIDNS	1483	1013	2029	1145	1075	1736

Table 3.4: Dobot fetch times

For a visual representation of the average fetching times of the Dobot for both wired and wireless setups, see the chart below.

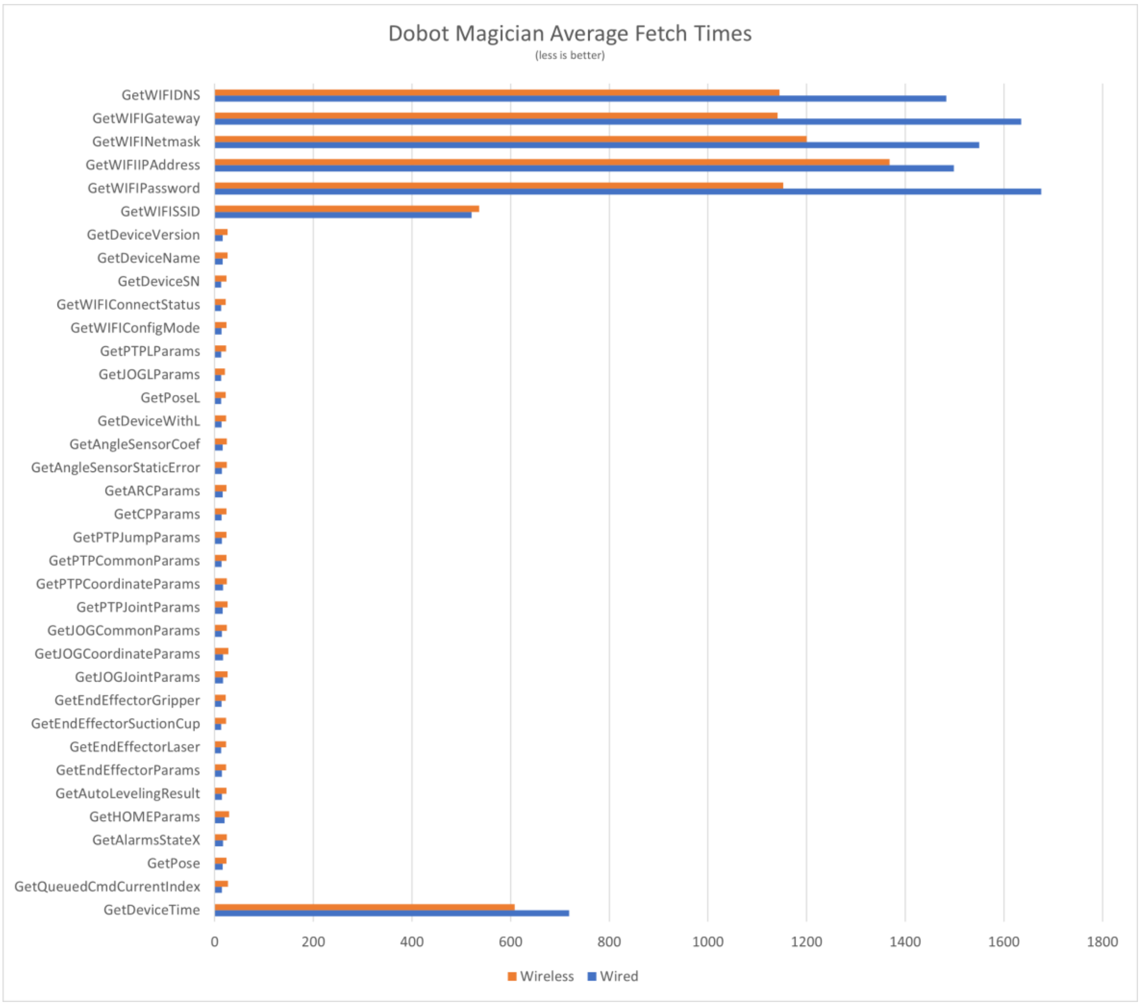


Image 3.1: Visualization of Dobot average fetch times

3.8.2 Jevois

Since the Jevois is directly connected to the host machine through a serial port, considering that timeout to the serial reading is set to 0 the consecutive fetch times, including reading, stripping and decoding the standardized message are on average around 1ms with a max fetch time of 4ms. The testing was done with Normal style 2D standardized messages. Fetch times on empty messages were not counted.

Chapter 4

Monitoring System: Prometheus

4.1 Introduction to Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit. Its rich feature list and solid architecture makes it a state-of-the-art solution for recording any purely numeric time series. When using traditional monitoring systems one can push the data onto the monitoring system for monitoring. With Prometheus however there is no such thing as pushing the data to it, as it instead scrapes (pulls) the data from specified endpoints on certain intervals and/or when it sees its needed, something that avoids overloading the system and provides reliability. Because of this even under failure conditions one can still see what monitored statistics are available. This characteristic makes Prometheus a great choice for this project since in an industrial environment there is a possibility for hardware failure/malfunction and one can easily rely on Prometheus to diagnose the problems [16].

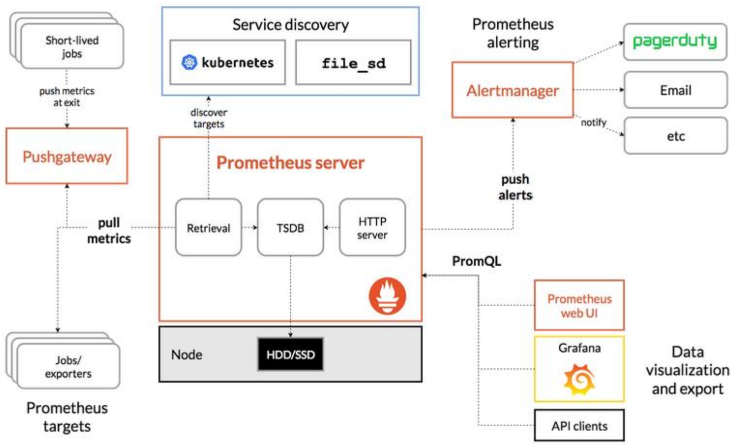


Image 4.1: Prometheus' architecture and its ecosystem components[16]

4.2 Python client library

For instrumenting the data exchange between the monitoring agent and the Prometheus server one can use the python client library (prometheus-client)[17]. The library provides 4 main metric types: Counter, Gauge, Histogram and Summary. Since the numeric values such as coordinates/location data can both increase and decrease, in the monitoring agent we will be using a Gauge for each metric that fluctuates. For general device information such as the device serial number the agent uses the Info which tracks key-value information about a whole target (once at the start of the routine). For alarms and object's identification, since they are not numeric values, the only way to approach monitoring such data is through Enum which is a type that supports strings as states and we can choose which state is enabled. For distinguishing metrics between different devices and/or other categorization we use labels. A label enables Prometheus' dimensional data model as any given combination of the labels for the same metric name identifies a particular instantiation of that metric. The supported device modules currently implement the device_id, device_type and station labels for querying based on specific device id (e.g. Dobot: 192.168.43.4), device type (e.g. Dobot, Jevois) and agent name, respectively. The latter can be used to create virtual stations (see 6.3). Since there is a great possibility that more than one of the same type of device monitors the same metric this approach helps with distinguishing which metric belongs to what device and synergizes well with Prometheus queries (PromQL)[18].

4.3 Hierarchical federation

For scaling the system horizontally one can add multiple monitoring stations with their respective monitoring agent's and Prometheus servers. In order to collect data from multiple monitoring stations we can use Prometheus' hierarchical federation. In the official documentation a federation is described as a way to allow a Prometheus server to scrape selected time series from another Prometheus server[25]. With this mechanism we can create a tree like structure with a main Prometheus server on the administration station being the root and scraping all the Prometheus servers on their respective monitoring station's, being the leaves. By doing so we can access the data flowing from all monitoring stations through one centralized Prometheus instance. For enabling such setup one can configure the main Prometheus instance to scrape from the /federate endpoints of the Prometheus server's running on the monitoring stations.

4.4 Long term storage

Prometheus stores metrics locally in a time series database for high efficiency[22]. This database is kept in memory and thus is not persisted. One might want these metrics to be stored for the long term in order to backup a history of metrics and/or analyze them for better insight. Prometheus writes incoming data to local storage and replicates it to remote storage in parallel. The "remote storage" as it is mentioned can exist on the localhost. In order to achieve that we can utilize Prometheus' capability to send and receive samples in a standardized format over HTTP. We can configure a Prometheus instance to write data to a remote database by adding a remote_write entry in its configuration file. One popular option for long term storage with Prometheus is VictoriaMetrics[23]. Victoria metrics can be used as a fast, cost-effective and scalable time series database which supports the Prometheus querying API and thus can be used with Grafana to visualize long term metrics. For a high-scale scenario this configuration can be applied to the main Prometheus server as described in the hierarchical federation, to store data from all the monitoring stations. VictoriaMetrics uses port 8428 by default and thus for using it for long term storage one must install and configure it to a host and add the following line[24] to the Prometheus configuration:

```
remote_write:
        - url: http://<victoriametrics-addr>:8428/api/v1/write
```

Chapter 5

Visualization: Grafana

5.1 Introduction to Grafana

Grafana is an open-source analytics platform for metrics. It allows for query, visualization and alert for better utilizing the metrics, in this case, provided by Prometheus[19]. Grafana is feature-rich and provides out-of-the-box support for Prometheus[20] which makes it a great option for this system.

5.2 Customization options

For customizing the visualization of the metrics one can create dashboards that display data retrieved through queries and more specifically PromQL, a query language designed for Prometheus timeseries metrics. The visualization is flexible as one can choose the type of visualization such as graph, heatmap, table and more as well as customize the dashboard placements and details.

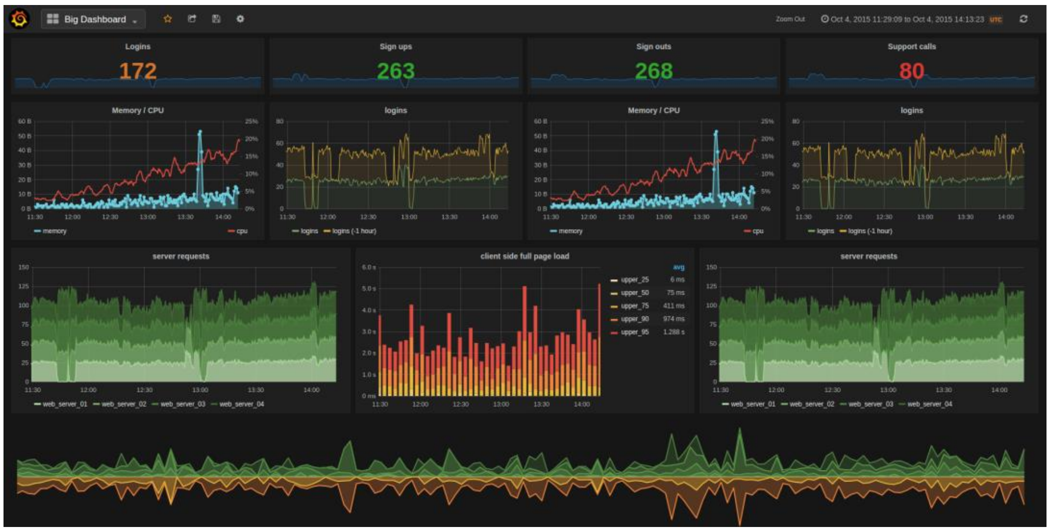


Image 5.1: Grafana dashboard example with a variety of visualization types[19]

Chapter 6

System Overview

6.1 High Level Architecture

The system consists of 3 main layers. From a bottom up approach these layers are the connectivity layer, the monitoring layer and the visualization layer. For each layer image the following apply: a solid line box indicates a device, a dotted box an interface and the line a connection. Labels in a line indicate the module used for establishing such connection. "Other Device" refers to a different device from the natively supported ones.

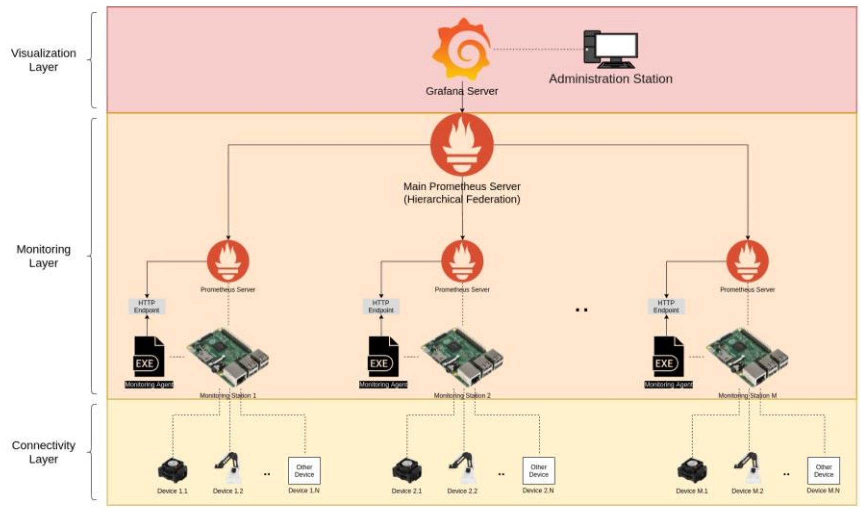


Image 6.1: Layer representation through abstract architecture

6.1.1 Connectivity Layer

The connectivity layer exists in the edge and is about the hardware stack for connecting the devices with the monitoring stations. Each device has its specific stack(s). The following diagram includes details regarding the natively supported devices. Both devices can utilize the serial ports on the monitoring agent to establish a connection but only the Dobot Magician can connect wirelessly (WLAN). The communication protocols needed for establishing a wireless communication with a Dobot Magician are the following (bottom-up): MAC, IPv4, UDP, Dobot Communication Protocol [7].

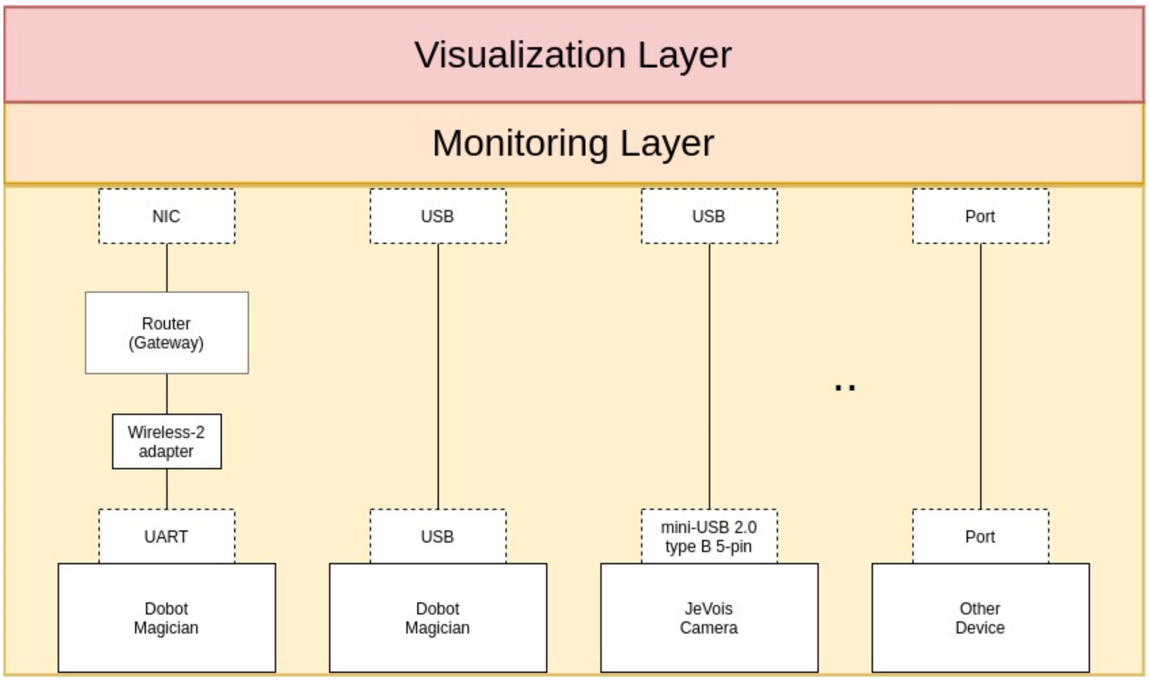


Image 6.2: Connectivity Layer

6.1.2 Monitoring Layer

This layer consists of one or more monitoring stations. Each monitoring station runs one monitoring agent and a local Prometheus server. The monitoring agent (agent.py) uses device_modules.py to connect to, fetch and disconnect from the devices. Each device has a different software (dependency) stack. For setups that involve more than one monitoring stations, a main Prometheus server configured as hierarchical federation is used as a central point for all metrics from all stations.

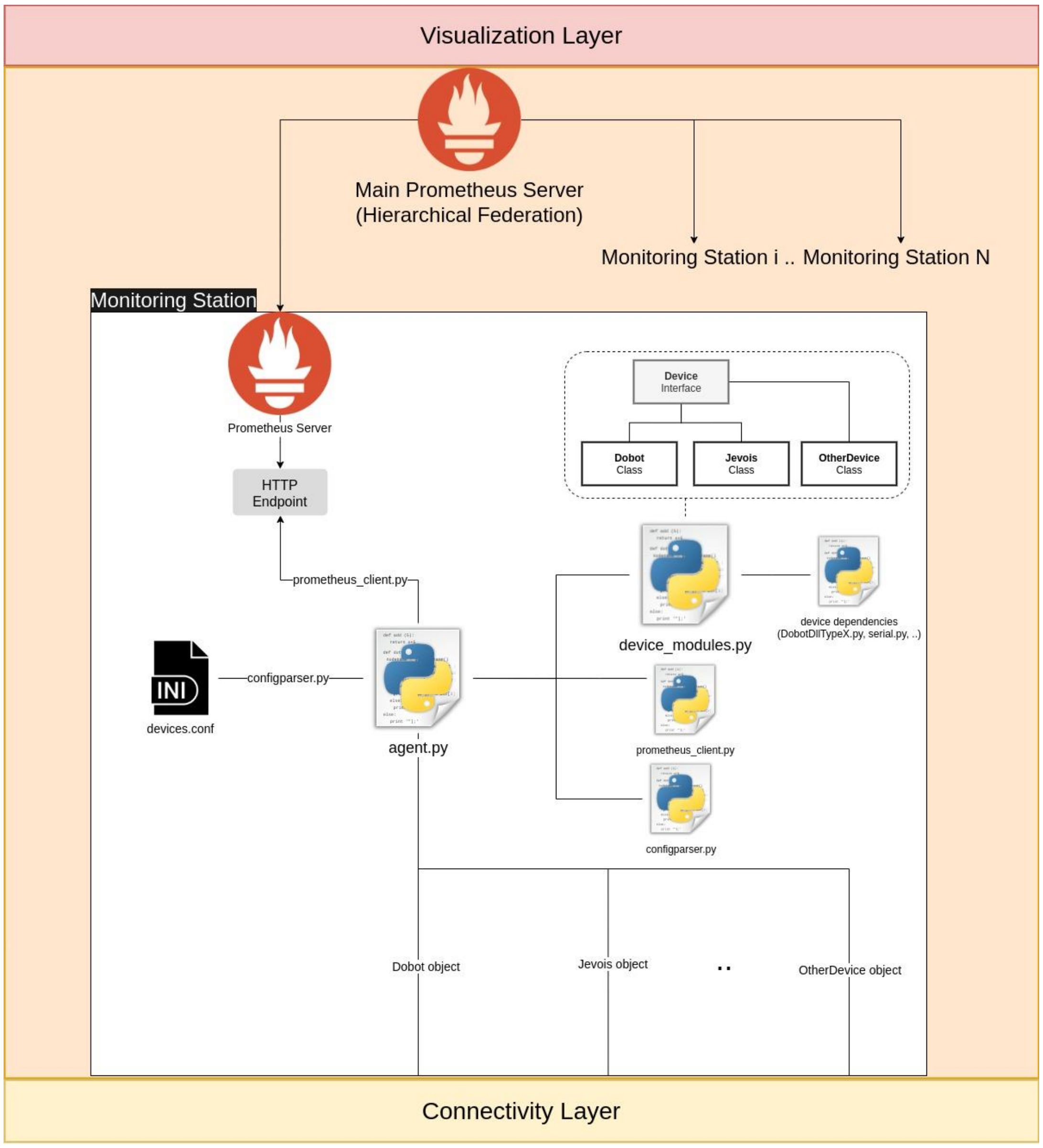


Image 6.3: Monitoring Layer

6.1.3 Visualization Layer

The visualization layer aims at providing a graphical interface for representing all fetched metrics and thus provide insight on the monitored devices. Any visualization software compatible with Prometheus can be used by using the main Prometheus server as a data source. The usual approach is having one administration station to act as a single monitoring interface however in more complex setups one can add multiple administration stations, for better separation of concerns.

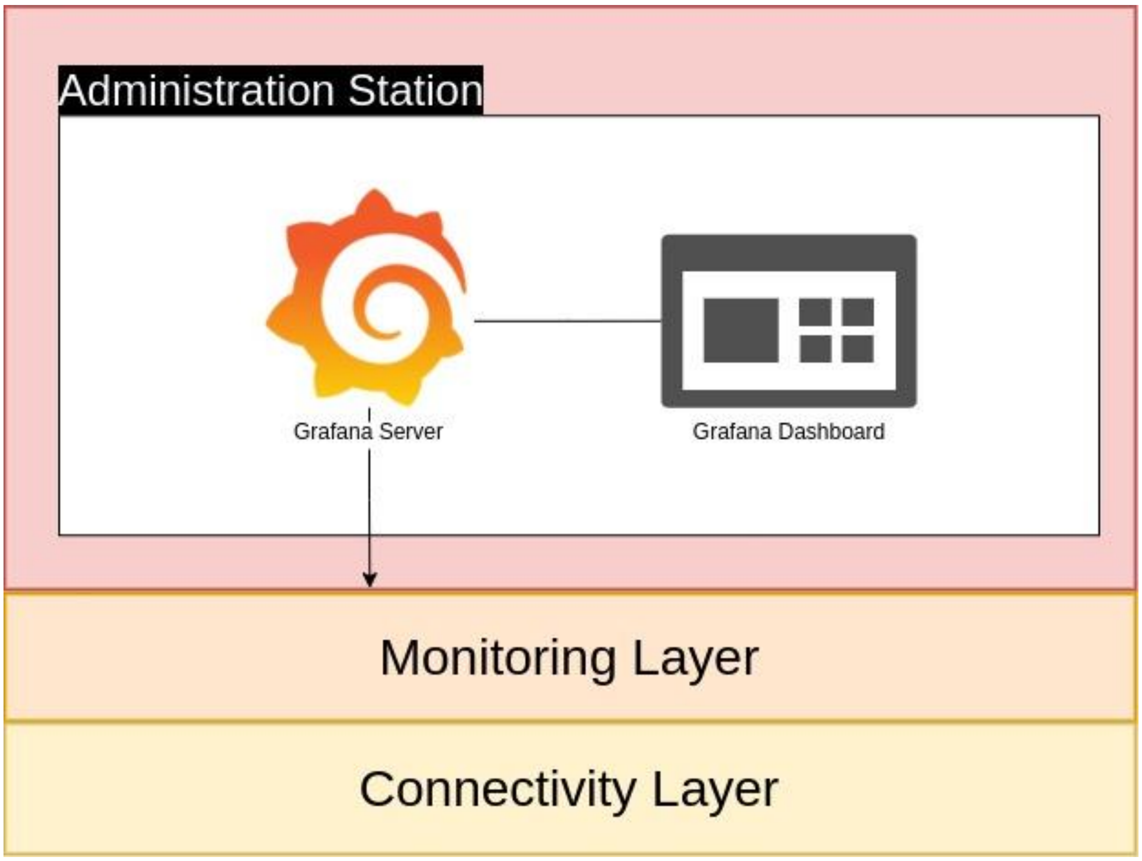


Image 6.4: Visualization Layer

6.1.4 Overall View

By combining the layers we have a complete flow of information from the device to the administrator's screen(s).

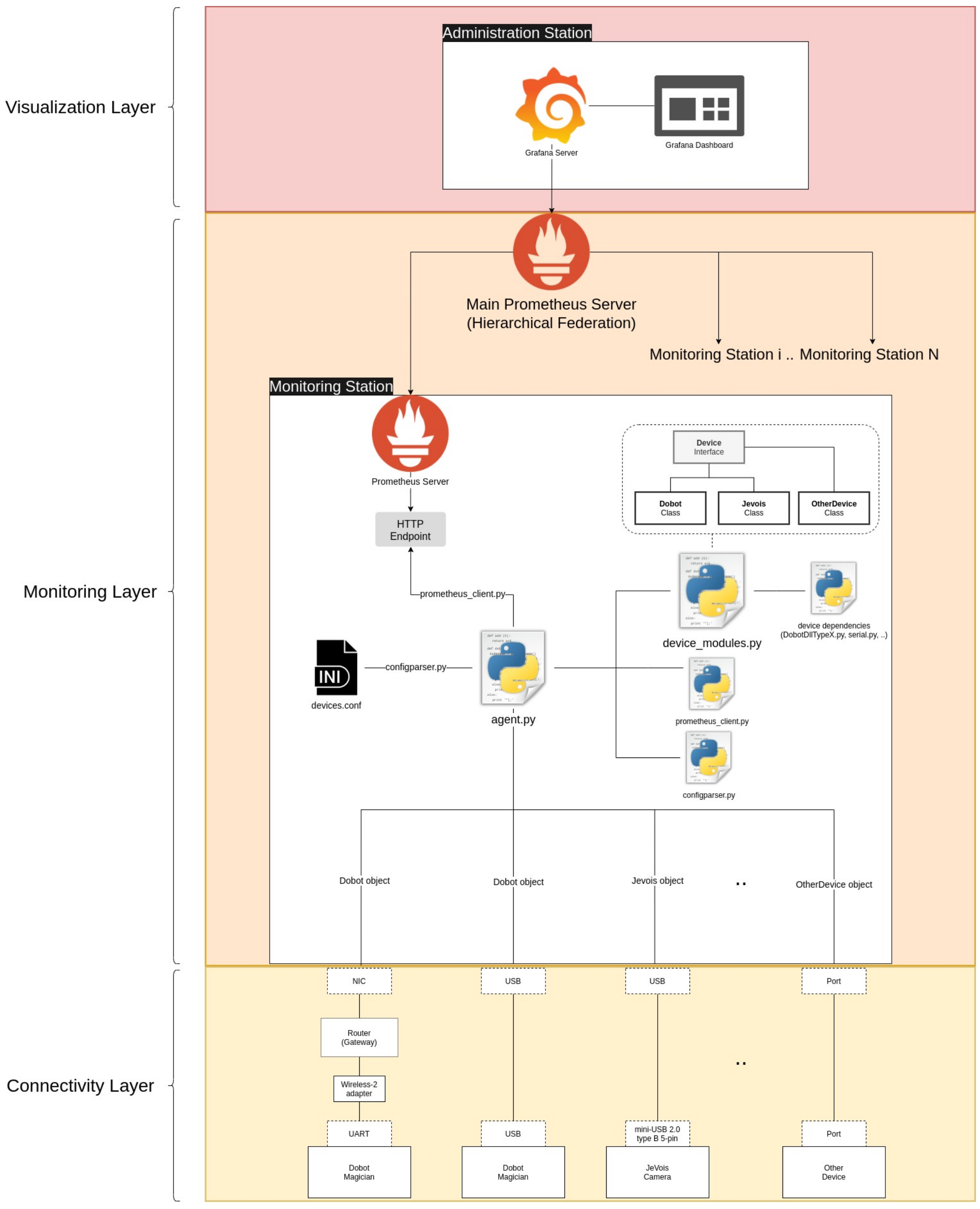


Image 6.5: Overall Layer View

6.2 Components

6.2.1 Hardware

In terms of hardware, host computer(s) acting as a monitoring/administration station(s) is needed. Also, the devices themselves that one needs to monitor (such as Dobot Magician, JeVois camera or other). If the connection between the host computer, running the monitoring agent, and a monitored device is not wireless all the necessary usb to serial port cables to establish the connection between the devices.

6.2.2 Software

The two open-source tools that are being utilized to form a complete monitoring system is Prometheus and Grafana (and their respective servers) which have been discussed in Chapter 4 and Chapter 5 respectively. The connecting piece of software between Prometheus and the physical devices is the monitoring agent (agent.py) and its dependencies (see 3.2).

6.3 Load Balancing

Monitoring stations can be used to group a set of devices (probably in the same section in the workflow) where the agent/station name option becomes relevant. Monitoring/grouping many devices through a single monitoring station (host computer) can bottleneck it and affect monitoring rates. In this case, another use of the agent’s name attribute is naming two or more stations with the same name thus creating a single virtual station. Since each metric supports the station label one can query based on the station name and get metrics from multiple stations while looking like it is one. For setups where the use of multiple monitoring stations is doable, one can use more than one monitoring station for monitoring a section and thus if distributed properly load balance the monitoring stations while still looking like a unified station.

Chapter 7

Conclusion

7.1 Conclusion

The task of monitoring comes with a lot of difficulties as devices might not provide the means to acquire the attributes one wishes to monitor, or it does it in an inefficient way. Monitoring adds overhead which needs to be properly calibrated in order to avoid any side-effects with the normal operations of the devices. However, if done correctly it can lead to great benefits for ensuring that everything is working as expected and to take proper (also could be automated) action in case they are not. With a monitoring history one can analyze the metrics further to optimize the environment and potentially predict any future failures or other useful conclusions regarding the work environment.

7.2 Possible extensions

Different device modules can be implemented to extend the agent's device support. An idea that arose but there was not enough time to implement, and test is the idea of general device modules e.g. Camera which would provide monitoring options that are common for all cameras such as FPS. Furthermore, all cameras with no means to get other attributes (e.g. standardized messages) could be connected through that module in a plug and play fashion. In contrast, device modules for cameras like JeVois could inherit from this base class to also include such information. Also give the ability to the agent to auto discover devices (by indicating the device type but not the port) by scanning all available serial and network ports, as well as reload the devices list dynamically to start monitoring new devices without the need to restart the agent.

References

[1] Hugh Boyes, Bil Hallaq, Joe Cunningham and Tim Watson, "The industrial internet of things (IIoT): An analysis framework", pp. 3, 2018

[2] In Lee and Kyoochun Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises", Kelley School of Business, Indiana University, pp. 1, 2015

[3] L. Aberle, "A comprehensive Guide to Enterprise IoT Project Success", IoT Agenda, pp. 1, 2015

[4] Dobot Magician Website, https://www.dobot.cc/dobot-magician/product-overview.html

[5] Conveyor Belt Kit, https://www.dobot.cc/products/conveyor-belt-kit-overview.html

[6] Dobot Magician API, https://download.dobot.cc/product-manual/dobot-magician/pdf/en/Dobot-Magician-API-DescriptionV1.2.3.pdf

[7] Dobot Communication Protocol, https://download.dobot.cc/product-manual/dobot-magician/pdf/en/Dobot-Communication-Protocol-V1.1.5.pdf

[8] Dobot Magician Demo Description, https://download.dobot.cc/development-protocol/dobot-magician/win7-win10/Demo/dobotdemo2.2/en/Dobot-Demo-V2.2-en.zip

[9] Magician Studio (Windows), https://download.dobot.cc/control-software/dobot-magician/win7-win10/1.9.4/DobotStudio(Windows)V1.9.4.zip

[10] Dobot Magician User Manual, https://download.dobot.cc/product-manual/dobot-magician/pdf/V1.7.0/en/Dobot-Magician-User-Guide-V1.7.0.pdf

[11] JeVois Smart Machine Vision Camera, http://www.jevois.org/

[12] JeVois: standardized serial messages formatting, http://jevois.org/doc/UserSerialStyle.html

[13] Connecting JeVois to Power and Data, http://jevois.org/doc/UserConnect.html

[14] Format of the .ini File, https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms717987(v=vs.85)

[15] configparser - Configuration file parser, https://docs.python.org/3/library/configparser.html
[16] Prometheus Overview, https://prometheus.io/docs/introduction/overview/
[17] Prometheus Python Client, https://github.com/prometheus/client_python
[18] Prometheus Data Model, https://prometheus.io/docs/concepts/data_model/
[19] Grafana, https://grafana.com/
[20] Getting started with Grafana and Prometheus, https://grafana.com/docs/grafana/latest/getting-started/getting-started-prometheus/
[21] Dobot ALARM, http://www.dobot.it/wp-content/uploads/2018/03/dobot-magician-alarm-en.pdf
[22] Prometheus: Storage, https://prometheus.io/docs/prometheus/latest/storage/
[23] VictoriaMetrics, https://victoriametrics.github.io/
[24] VictoriaMetrics: Prometheus Setup, https://victoriametrics.github.io/#prometheus-setup
[25] Prometheus: Federation, https://prometheus.io/docs/prometheus/latest/federation/

Note: All online references were accessed on the date of the publishing of this paper.
Github repository of the project: https://github.com/akomis/diploma-project

Appendix A

List of abbreviations and their meaning used in this paper.

API: Application Programming Interface

DNS: Domain Name System

FPS: Frames Per Second

GUI: Graphical User Interface

HTTP: HyperText Transfer Protocol

IIOT: Industrial Internet of Things

IOT: Internet of Things

IP: Internet Protocol

MAC: Medium Access Control

NIC: Network Interface Controller

PQL/PromQL: Prometheus Query Language

UART: Universal Asynchronous Receiver/Transmitter

UDP: User Datagram Protocol

WLAN: Wireless Local Area Network

Appendix B

Source code of the monitoring agent (agent.py)

```
import sys
import time
import argparse
import webbrowser
import configparser
from threading import Thread
from prometheus_client import start_http_server
from device_modules import *

class Agent():
    termcolors = {
    "B":"\033[1m","U":"\033[4m","H":"\033[95m",
    "OK":"\033[92m","INFO":"\033[94m",
    "WARNING":"\033[93m","ERROR":"\033[91m",
    "END":"\033[0m"}

    def __init__(self, devicesFilename:str, name:str, prometheusPort:int, killSwitch:bool, verbose:bool,
color:bool):
        self.stopped = False
        self.config = configparser.ConfigParser()
        self.devicesFilename = devicesFilename
        self.name = name
        self.prometheusPort = prometheusPort
        self.killSwitch = killSwitch
        self.verbose = verbose
        if not color:
            for attr in Agent.termcolors:
                Agent.termcolors[attr] = ""

        self.__readConfig()
        self.validSections = self.__validateConfig()
        self.devices = []

    def agentPrint(self, s, type=""):
        termcolors = Agent.termcolors
        prefix = self.name + " (" + time.ctime() + "): "
        body = s + termcolors["END"]

        if type == "i":
            print(prefix + termcolors["INFO"] +  "[INFO] " + body)
        elif type == "o":
            print(prefix + termcolors["OK"] + "[OK] " + body)
        elif type == "w":
            print(prefix + termcolors["WARNING"] + "[WARNING] " + body)
        elif type == "e":
            print(prefix + termcolors["ERROR"] + "[ERROR] " + body, file=sys.stderr)
        elif type == "f":
            print(prefix + termcolors["ERROR"] + "[" + termcolors["U"] + "FATAL" + termcolors["END"] +
termcolors["ERROR"] + "] " + body, file=sys.stderr)
        else:
            print(prefix + s)

    def __readConfig(self):
        try:
            check = self.config.read(self.devicesFilename)

            if len(check) == 0:
                raise Exception("Couldn't find file")
        except Exception as e:
            self.agentPrint("Can't read configuration file \"" + self.devicesFilename + "\" (" + str(e) + ")",
type="e")
            exit(3)

        self.agentPrint("Succesfully opened configuration file \"" + self.devicesFilename + "\"", type="o")

    def __validateConfig(self):
        self.agentPrint("Validating configuration file..", type="i")

        validBooleanValues = ["1","yes","true","on","0","no","false","off"]
        validSections = {}

        # Check configuration validity for each defined devices
        flag = False
        for sectionName in self.config.sections():
            try:
```

```
            part = sectionName.split(":")

            if len(part) != 2:
                raise Exception("\"" + sectionName + " is not a valid device entry. All device entries should
follow this format [DEVICE_TYPE:PORT]")

            deviceType = part[0]
            connectionPort = part[1]

            if deviceType not in globals():
                raise Exception("The agent does not support the \"" + deviceType + "\" device type. Make
sure the appropriate device module exists in device_module.py (case-sensitive)")

            section = self.config[sectionName]
            entityClass = globals()[deviceType]

            if entityClass.options == Device.options or len(entityClass.options) == 0:
                raise Exception("Cannot validate \"" + deviceType + "\". Make sure the options{} dictionary
is implemented.")
        except Exception as e:
            flag = True
            self.agentPrint(str(e), type="e")
            self.agentPrint("\"" + sectionName + "\" device will not be monitored.", type="w")
            continue

        # Check if device fields in configuration are valid (supported by module)
        options = {}
        options.update(entityClass.options)
        options.update(Device.options)
        errorCount = 0
        for option in section:
            try:
                if option not in options:
                    raise Exception("\"" + option + "\" is not a valid option for section \"" + sectionName +
"\".")

                configValue = section[option]
                optionsValue = options[option]

                # Check if value type is correct
```

```
                    if isinstance(optionsValue, bool) and configValue not in validBooleanValues:
                        raise Exception("Value \"" + configValue + "\" for option \"" + option + "\" in section \"" +
sectionName +"\" is not valid (can only be 1|yes|true|on|0|no|false|off)")

                    try:
                        type(optionsValue)(configValue)
                    except:
                        raise Exception("Value \"" + configValue + "\" for option \"" + option + "\" in section \"" +
sectionName +"\" is not valid (must be of type " + str(type(optionsValue).__name__) + ").")
                except Exception as e:
                    flag = True
                    errorCount += 1
                    self.agentPrint(str(e), type="e")

            if errorCount > 0:
                self.agentPrint(str(errorCount) + " error(s) in \"" + sectionName + "\" section. The device will
not be monitored. Please resolve the errors in order for this device to be monitored.", type="w")
            else:
                try:
                    device = entityClass(section, connectionPort, self.name)
                except Exception as e:
                    flag = True
                    self.agentPrint("\"" + deviceType + "\" device module does not properly implement the
Device interface", type="e")
                    self.agentPrint("\"" + sectionName + "\" device will not be monitored.", type="w")
                    continue

                if device.activeAttr == 0:
                    flag = True
                    self.agentPrint("Device \"" + sectionName + "\" has 0 enabled attributes to be monitored",
type="e")
                    self.agentPrint("\"" + sectionName + "\" device will not be monitored.", type="w")
                    continue

                validSections[sectionName] = device

        if flag:
            self.agentPrint("For more information use --more.", type="i")
            if self.killSwitch:
                self.agentPrint("Killswitch is enabled. Exiting..", type="f")
                exit(6)
        else:
            if self.verbose:
                self.agentPrint("No errors in \"" + self.devicesFilename + "\"", type="o")

        return validSections

    def __connectDevices(self):
        # Discover through the validated config which devices should be monitored
        for sectionName in self.validSections:
            start = time.time()
            device = self.validSections[sectionName]

            if self.verbose:
                self.agentPrint("Connecting to " + device.id + "..", type="i")

            try:
                device.connect()
                elapsed = time.time() - start

                self.devices.append(device)
                if self.verbose:
                    self.agentPrint("Device " + device.type + " at " + device.port + " connected succesfully! (" +
str(round(elapsed*1000)) + "ms)", type="o")
                else:
                    self.agentPrint("Device " + device.type + " at " + device.port + " connected succesfully!",
type="o")
            except Exception as e:
                self.agentPrint(device.type + " at " + device.port + " cannot be connected. (" + str(e) + ")",
type="e")
                self.agentPrint("Device " + device.type + " at " + device.port + " will not be monitored.",
type="w")
                if self.killSwitch:
                    self.agentPrint("Killswitch is enabled. Exiting..", type="f")
                    exit(7)

    def __fetchFrom(self, device):
        while not self.stopped:
            try:
                start = time.time()
                device.fetch()
                elapsed = time.time() - start
                if self.verbose:
                    self.agentPrint("Fetched from " + device.id + " in " + str(round(elapsed*1000)) + " ms",
type="o")

                time.sleep(device.timeout / 1000)
            except Exception as e:
                self.agentPrint("Couldn't fetch from " + device.id + " (" + str(e) + ")", type="e")
                time.sleep(device.timeout / 1000)

    def startRoutine(self):
        self.agentPrint("Connecting to devices listed in \"" + self.devicesFilename + "\"..", type="i")
        self.__connectDevices()

        if len(self.devices) == 0:
            self.agentPrint("No devices connected to the agent. Exiting..", type="f")
            sys.exit(8)

        self.agentPrint("Starting prometheus server at port " + str(self.prometheusPort) + "..", type="i")
        start_http_server(self.prometheusPort)

        try:
            threads = {}
            for device in self.devices:
                threads[device] = Thread(target = self.__fetchFrom, args=(device,))

            for device in threads:
                threads[device].start()
                if self.verbose:
                    self.agentPrint("Started monitoring for device " + device.id + " with " + str(device.activeAttr)
+ " active attributes (fetching timeout: " + str(device.timeout) + "ms)", type="o")

            if not self.verbose:
                self.agentPrint("Monitoring..", type="i")

            while 1:
                pass
        except KeyboardInterrupt:
            self.stopped = True
            self.agentPrint("Waiting for active fetching to complete..", type="i")
            for device in threads:
                threads[device].join()

            self.agentPrint("Disconnecting devices..", type="i")
            for device in self.devices:
                device.disconnect()

            if self.verbose:
                self.agentPrint("Disconnected devices", type="o")

            exit(0)

def isPort(value):
    port = 0
    try:
        port = int(value)
        if port > 65535 or port < 0:
            raise Exception()
    except Exception as e:
        raise argparse.ArgumentTypeError("\"%s\" is not a valid port number (must be an integer between 1
and 65535)" % value)

    return port

def main():
    parser = argparse.ArgumentParser(description="Agent Settings")
    parser.add_argument("-d", "--devices", default="devices.conf", help="specify discovery/configuration
file absolute path (default: \".\\devices.conf\")")
    parser.add_argument("-n", "--name", default="Agent0", help="specify symbolic agent/station name
used for separation/grouping of stations (default: \"Agent0\")")
    parser.add_argument("-p", "--promport", type=isPort, default=8000, help="specify port number for the
Prometheus endpoint (default: 8000)")
    parser.add_argument("-k", "--killswitch", action="store_true", help="exit agent if error occurs in
validation/connection phase")
    parser.add_argument("-v", "--verbose", action="store_true", help="print actions with details in
standard output")
    parser.add_argument("-c", "--color", action="store_true", help="print color rich messages to terminal
(terminal needs to support ANSI escape colors)")
```

```
    parser.add_argument("-m", "--more", action="store_true", help="open README.md with
configuration and implementation details and exit")
    args = parser.parse_args()

    if args.more:
        webbrowser.open("..\README.md")
        exit(0)

    Agent(args.devices, args.name, args.promport, args.killswitch, args.verbose, args.color).startRoutine()

if __name__ == "__main__":
    main()
```

Appendix C

Source code of device modules used by the monitoring agent (device_modules.py)

```
from abc import ABC, abstractmethod
from prometheus_client import Info, Gauge, Enum
import runtime.DobotDllTypeX as dTypeX
import serial

class Device(ABC):
    options = {"timeout":0} # Default device options/attributes

    def __init__(self, config_section, port, host):
        self.section = config_section
        self.port = port
        self.host = host
        self.type = type(self).__name__
        self.id = self.type + ":" + self.port

        self.timeout = self.section.getint("timeout", fallback=Device.options["timeout"])
        if self.timeout < Device.options["timeout"]: self.timeout = Device.options["timeout"]

        activeCounter = 0
        for key in type(self).options:
            if isinstance(type(self).options[key], bool) and self.isEnabled(key):
                activeCounter += 1

        self.activeAttr = activeCounter

    @abstractmethod
    def connect(self):
        pass

    @abstractmethod
    def fetch(self):
        pass
    @abstractmethod
    def disconnect(self):
        pass

    def isEnabled(self, attr):
        if not isinstance(type(self).options[attr], bool):
            raise Exception("\"" + attr + "\" attribute is not a monitoring option.")
        return self.section.getboolean(attr, fallback=type(self).options[attr])

    def isCallEnabled(self, attrList):
        for attr in attrList:
            if self.isEnabled(attr):
                return True

        return False

class Dobot(Device):
    options =
{"devicesn":True,"devicename":True,"deviceversion":True,"devicetime":False,"queueindex":False,

"posex":True,"posey":True,"posez":True,"poser":True,"anglebase":True,"anglereararm":True,"angleforearm":
True,
    "angleendeffector":True,"alarmsstate":True,"homex":False,"homey":False,"homez":False,"homer":False,

"endeffectorx":False,"endeffectory":False,"endeffectorz":False,"laserstatus":False,"suctioncupstatus":False,"g
ripperstatus":False,"jogbasevelocity":False,

"jogreararmvelocity":False,"jogforearmvelocity":False,"jogendeffectorvelocity":False,"jogbaseacceleration":
False,"jogreararmacceleration":False,

"jogforearmacceleration":False,"jogendeffectoracceleration":False,"jogaxisxvelocity":False,"jogaxisyvelocity
":False,"jogaxiszvelocity":False,

"jogaxisrvelocity":False,"jogaxisxacceleration":False,"jogaxisyacceleration":False,"jogaxiszacceleration":Fals
e,"jogaxisracceleration":False,
    "jogvelocityratio":False,"jogaccelerationratio":False,"ptpbasevelocity":False,"ptpreararmvelocity":False,
"ptpforearmvelocity":False,"ptpendeffectorvelocity":False,"ptpbaseacceleration":False,"ptprearacceleratio
n":False,
    "ptpforearmacceleration":False,"ptpendeffectoracceleration":False,"ptpaxisxyzvelocity":False,

"ptpaxisrvelocity":False,"ptpaxisxyzacceleration":False,"ptpaxisracceleration":False,"ptpvelocityratio":False,
    "ptpaccelerationratio":False,"liftingheight":False,"heightlimit":False,
    "cpvelocity":False,"cpacceleration":False,"arcxyzvelocity":False,"arcrvelocity":False,
    "arcxyzacceleration":False,"arcracceleration":False,"anglestaticerrrear":False,
    "anglestaticerrfront":False,"anglecoefrear":False,"anglecoeffront":False,"slidingrailstatus":False,
    "slidingrailpose":False,"slidingrailjogvelocity":False,"slidingrailjogacceleration":False,
    "slidingrailptpvelocity":False,"slidingrailptpacceleration":False,"wifimodulestatus":False,
    "wificonnectionstatus":False,"wifissid":False,"wifipassword":False,"wifiipaddress":False,
    "wifinetmask":False,"wifigateway":False,"wifidns":False}

    deviceInfo = Info("dobot_magician", "General information about monitored Dobot Magician device",
["device_id","device_type","station"])
    wifiInfo = Info("wifi", "Information regarding the device's wifi connection",
["device_id","device_type","station"])
    deviceTime = Gauge("device_time","Device's clock/time", ["device_id","device_type","station"])
    queueIndex = Gauge("queue_index","Current index in command queue",
["device_id","device_type","station"])
    poseX = Gauge("pose_x","Real-time cartesian coordinate of device's X axis",
["device_id","device_type","station"])
    poseY = Gauge("pose_y","Real-time cartesian coordinate of device's Y axis",
["device_id","device_type","station"])
    poseZ = Gauge("pose_z","Real-time cartesian coordinate of device's Z axis",
["device_id","device_type","station"])
    poseR = Gauge("pose_r","Real-time cartesian coordinate of device's R axis",
["device_id","device_type","station"])
    angleBase = Gauge("angle_base","Base joint angle", ["device_id","device_type","station"])
    angleRearArm = Gauge("angle_rear_arm","Rear arm joint angle", ["device_id","device_type","station"])
    angleForearm = Gauge("angle_forearm","Forearm joint angle", ["device_id","device_type","station"])
    angleEndEffector = Gauge("angle_end_effector","End effector joint angle",
["device_id","device_type","station"])
    alarmsState = Enum("alarms_state", "Device alarms state", ["device_id","device_type","station"],
states=dTypeX.alarmStates)
```

```
    homeX = Gauge("home_x","Home position for X axis", ["device_id","device_type","station"])
    homeY = Gauge("home_y","Home position for Y axis", ["device_id","device_type","station"])
    homeZ = Gauge("home_z","Home position for Z axis", ["device_id","device_type","station"])
    homeR = Gauge("home_r","Home position for R axis", ["device_id","device_type","station"])
    endEffectorX = Gauge("end_effector_x","X-axis offset of end effector",
["device_id","device_type","station"])
    endEffectorY = Gauge("end_effector_y","Y-axis offset of end effector",
["device_id","device_type","station"])
    endEffectorZ = Gauge("end_effector_z","Z-axis offset of end effector",
["device_id","device_type","station"])
    laserStatus = Enum("laser_status","Status (enabled/disabled) of laser",
["device_id","device_type","station"], states=["enabled","disabled"])
    suctionCupStatus = Enum("suction_cup_status","Status (enabled/disabled) of suction cup",
["device_id","device_type","station"], states=["enabled","disabled"])
    gripperStatus = Enum("gripper_status","Status (enabled/disabled) of gripper",
["device_id","device_type","station"], states=["enabled","disabled"])
    jogBaseVelocity = Gauge("jog_base_velocity","Velocity (°/s) of base joint in jogging mode",
["device_id","device_type","station"])
    jogRearArmVelocity = Gauge("jog_rear_arm_velocity","Velocity (°/s) of rear arm joint in jogging mode",
["device_id","device_type","station"])
    jogForearmVelocity = Gauge("jog_forearm_velocity","Velocity (°/s) of forearm joint in jogging mode",
["device_id","device_type","station"])
    jogEndEffectorVelocity = Gauge("jog_end_effector_velocity","Velocity (°/s) of end effector joint in
jogging mode", ["device_id","device_type","station"])
    jogBaseAcceleration = Gauge("jog_base_acceleration","Acceleration (°/s^2) of base joint in jogging
mode", ["device_id","device_type","station"])
    jogRearArmAcceleration = Gauge("jog_rear_arm_acceleration","Acceleration (°/s^2) of rear arm joint in
jogging mode", ["device_id","device_type","station"])
    jogForearmAcceleration = Gauge("jog_forearm_acceleration","Acceleration (°/s^2) of forearm joint in
jogging mode", ["device_id","device_type","station"])
    jogEndEffectorAcceleration = Gauge("jog_end_effector_acceleration","Acceleration (°/s^2) of end effector
joint in jogging mode", ["device_id","device_type","station"])
    jogAxisXVelocity = Gauge("jog_axis_x_velocity","Velocity (mm/s) of device's X axis (cartesian
coordinate) in jogging mode", ["device_id","device_type","station"])
    jogAxisYVelocity = Gauge("jog_axis_y_velocity","Velocity (mm/s) of device's Y axis (cartesian
coordinate) in jogging mode", ["device_id","device_type","station"])
```

```
    jogAxisZVelocity = Gauge("jog_axis_z_velocity","Velocity (mm/s) of device's Z axis (cartesian
coordinate) in jogging mode", ["device_id","device_type","station"])
    jogAxisRVelocity = Gauge("jog_axis_r_velocity","Velocity (mm/s) of device's R axis (cartesian
coordinate) in jogging mode", ["device_id","device_type","station"])
    jogAxisXAcceleration = Gauge("jog_axis_x_acceleration","Acceleration (mm/s^2) of device's X axis
(cartesian coordinate) in jogging mode", ["device_id","device_type","station"])
    jogAxisYAcceleration = Gauge("jog_axis_y_acceleration","Acceleration (mm/s^2) of device's Y axis
(cartesian coordinate) in jogging mode", ["device_id","device_type","station"])
    jogAxisZAcceleration = Gauge("jog_axis_z_acceleration","Acceleration (mm/s^2) of device's Z axis
(cartesian coordinate) in jogging mode", ["device_id","device_type","station"])
    jogAxisRAcceleration = Gauge("jog_axis_r_acceleration","Acceleration (mm/s^2) of device's R axis
(cartesian coordinate) in jogging mode", ["device_id","device_type","station"])
    jogVelocityRatio = Gauge("jog_velocity_ratio","Velocity ratio of all axis (joint and cartesian coordinate
system) in jogging mode", ["device_id","device_type","station"])
    jogAccelerationRatio = Gauge("jog_acceleration_ratio","Acceleration ratio of all axis (joint and cartesian
coordinate system) in jogging mode", ["device_id","device_type","station"])
    ptpBaseVelocity = Gauge("ptp_base_velocity","Velocity (°/s) of base joint in point to point mode",
["device_id","device_type","station"])
    ptpRearArmVelocity = Gauge("ptp_rear_arm_velocity","Velocity (°/s) of rear arm joint in point to point
mode", ["device_id","device_type","station"])
    ptpForearmVelocity = Gauge("ptp_forearm_velocity","Velocity (°/s) of forearm joint in point to point
mode", ["device_id","device_type","station"])
    ptpEndEffectorVelocity = Gauge("ptp_end_effector_velocity","Velocity (°/s) of end effector joint in point
to point mode", ["device_id","device_type","station"])
    ptpBaseAcceleration = Gauge("ptp_base_acceleration","Acceleration (°/s^2) of base joint in point to point
mode", ["device_id","device_type","station"])
    ptpRearArmAcceleration = Gauge("ptp_rear_arm_acceleration","Acceleration (°/s^2) of rear arm joint in
point to point mode", ["device_id","device_type","station"])
    ptpForearmAcceleration = Gauge("ptp_forearm_acceleration","Acceleration (°/s^2) of forearm joint in
point to point mode", ["device_id","device_type","station"])
    ptpEndEffectorAcceleration = Gauge("ptp_end_effector_acceleration","Acceleration (°/s^2) of end effector
joint in point to point mode", ["device_id","device_type","station"])
    ptpAxisXYZVelocity = Gauge("ptp_axis_xyz_velocity","Velocity (mm/s) of device's X, Y, Z axis
(cartesian coordinate) in point to point mode", ["device_id","device_type","station"])
    ptpAxisRVelocity = Gauge("ptp_axis_r_velocity","Velocity (mm/s) of device's R axis (cartesian
coordinate) in point to point mode", ["device_id","device_type","station"])
```

```
    ptpAxisXYZAcceleration = Gauge("ptp_axis_x_y_z_acceleration","Acceleration (mm/s^2) of device's X,
Y, Z axis (cartesian coordinate) in point to point mode", ["device_id","device_type","station"])
    ptpAxisRAcceleration = Gauge("ptp_axis_r_acceleration","Acceleration (mm/s^2) of device's R axis
(cartesian coordinate) in point to point mode", ["device_id","device_type","station"])
    ptpVelocityRatio = Gauge("ptp_velocity_ratio","Velocity ratio of all axis (joint and cartesian coordinate
system) in point to point mode", ["device_id","device_type","station"])
    ptpAccelerationRatio = Gauge("ptp_acceleration_ratio","Acceleration ratio of all axis (joint and cartesian
coordinate system) in point to point mode", ["device_id","device_type","station"])
    liftingHeight = Gauge("lifting_height","Lifting height in jump mode",
["device_id","device_type","station"])
    heightLimit = Gauge("height_limit","Max lifting height in jump mode",
["device_id","device_type","station"])
    cpVelocity = Gauge("cp_velocity","Velocity (mm/s) in cp mode", ["device_id","device_type","station"])
    cpAcceleration = Gauge("cp_acceleration","Acceleration (mm/s^2) in cp mode",
["device_id","device_type","station"])
    arcXYZVelocity = Gauge("arc_x_y_z_velocity","Velocity (mm/s) of X, Y, Z axis in arc mode",
["device_id","device_type","station"])
    arcRVelocity = Gauge("arc_r_velocity","Velocity (mm/s) of R axis in arc mode",
["device_id","device_type","station"])
    arcXYZAcceleration = Gauge("arc_x_y_z_acceleration","Acceleration (mm/s^2) of X, Y, Z axis in arc
mode", ["device_id","device_type","station"])
    arcRAcceleration = Gauge("arc_r_acceleration","Acceleration (mm/s^2) of R axis in arc mode",
["device_id","device_type","station"])
    angleStaticErrRear = Gauge("angle_static_err_rear","Rear arm angle sensor static error",
["device_id","device_type","station"])
    angleStaticErrFront = Gauge("arc_static_err_front","Forearm angle sensor static error",
["device_id","device_type","station"])
    angleCoefRear = Gauge("angle_coef_rear","Rear arm angle sensor linearization parameter",
["device_id","device_type","station"])
    angleCoefFront = Gauge("angle_coef_front","Forearm angle sensor linearization parameter",
["device_id","device_type","station"])
    slidingRailStatus = Enum("sliding_rail_status","Sliding rail's status (enabled/disabled)",
["device_id","device_type","station"], states=["enabled","disabled"])
    slidingRailPose = Gauge("sliding_rail_pose","Sliding rail's real-time pose in mm",
["device_id","device_type","station"])
    slidingRailJogVelocity = Gauge("sliding_rail_jog_velocity","Velocity (mm/s) of sliding rail in jogging
mode", ["device_id","device_type","station"])
```

```
    slidingRailJogAcceleration = Gauge("sliding_rail_jog_acceleration","Acceleration (mm/s^2) of sliding rail
in jogging mode", ["device_id","device_type","station"])
    slidingRailPtpVelocity = Gauge("sliding_rail_ptp_velocity","Velocity (mm/s) of sliding rail in point to
point mode", ["device_id","device_type","station"])
    slidingRailPtpAcceleration = Gauge("sliding_rail_ptp_acceleration","Acceleration (mm/s^2) of sliding rail
in point to point mode", ["device_id","device_type","station"])
    wifiModuleStatus = Enum("wifi_module_status","Wifi module status (enabled/disabled)",
["device_id","device_type","station"], states=["enabled","disabled"])
    wifiConnectionStatus = Enum("wifi_connection_status","Wifi connection status (connected/not
connected)", ["device_id","device_type","station"], states=["enabled","disabled"])

    def connect(self):
        stateInfo = {1:"Not Found", 2:"Occupied"}

        try:
            self.api, state = dTypeX.ConnectDobotX(self.port)
            if state[0] == dTypeX.DobotConnect.DobotConnect_NoError:
                self.__initialize()
            else:
                raise Exception(stateInfo[state[0]])
        except Exception as e:
            raise Exception(str(e))

    def __initialize(self):
        enabledDeviceInfo = {}
        if self.isEnabled("devicesn"):
            enabledDeviceInfo["serial"] = dTypeX.GetDeviceSN(self.api)[0]
        if self.isEnabled("devicename"):
            enabledDeviceInfo["name"] = dTypeX.GetDeviceName(self.api)[0]
        if self.isEnabled("deviceversion"):
            enabledDeviceInfo["version"] = ".".join(list(map(str, dTypeX.GetDeviceVersion(self.api))))
        if len(enabledDeviceInfo) > 0:
            Dobot.deviceInfo.labels(device_id=self.id, device_type=self.type,
station=self.host).info(enabledDeviceInfo)

        enabledWifiInfo = {}
        if self.isEnabled("wifissid"):
            enabledWifiInfo["ssid"] = dTypeX.GetWIFISSID(self.api)[0]
        if self.isEnabled("wifipassword"):
            enabledWifiInfo["password"] = dTypeX.GetWIFIPassword(self.api)[0]
        if self.isEnabled("wifiipaddress"):
            enabledWifiInfo["ip_address"] = ".".join(list(map(str, dTypeX.GetWIFIIPAddress(self.api)[1:])))
        if self.isEnabled("wifinetmask"):
            enabledWifiInfo["netmask"] = ".".join(list(map(str, dTypeX.GetWIFINetmask(self.api))))
        if self.isEnabled("wifigateway"):
            enabledWifiInfo["gateway"] = ".".join(list(map(str, dTypeX.GetWIFIGateway(self.api))))
        if self.isEnabled("wifidns"):
            enabledWifiInfo["dns"] = ".".join(list(map(str, dTypeX.GetWIFIDNS(self.api))))
        if len(enabledWifiInfo) > 0:
            Dobot.wifiInfo.labels(device_id=self.id, device_type=self.type,
station=self.host).info(enabledWifiInfo)

        self.GetPose =
self.isCallEnabled(["posex","posey","posez","poser","anglebase","anglereararm","angleforearm","angleendef
fector"])
        self.GetHomeParams = self.isCallEnabled(["homex","homey","homez","homer"])
        self.GetEndEffectorParams = self.isCallEnabled(["endeffectorx","endeffectory","endeffectorz"])
        self.GetJOGGointParams =
self.isCallEnabled(["jogbasevelocity","jogreararmvelocity","jogforearmvelocity","jogendeffectorvelocity",
        "jogbaseacceleration","jogreararmacceleration","jogforearmacceleration","jogendeffectoracceleration"])
        self.GetJOGCoordinateParams =
self.isCallEnabled(["jogaxisxvelocity","jogaxisyvelocity","jogaxiszvelocity","jogaxisrvelocity",
        "jogaxisxacceleration","jogaxisyacceleration","jogaxiszacceleration","jogaxisracceleration"])
        self.GetJOGCommonParams = self.isCallEnabled(["jogvelocityratio","jogaccelerationratio"])
        self.GetPTPJointParams =
self.isCallEnabled(["ptpbasevelocity","ptpreararmvelocity","ptpforearmvelocity","ptpendeffectorvelocity",
        "ptpbaseacceleration","ptpreararmacceleration","ptpforearmacceleration","ptpendeffectoracceleration"])
        self.GetPTPCoordinateParams =
self.isCallEnabled(["ptpaxisxyzvelocity","ptpaxisrvelocity","ptpaxisxyzacceleration","ptpaxisracceleration"])
        self.GetPTPCommonParams = self.isCallEnabled(["ptpvelocityratio","ptpaccelerationratio"])
        self.GetPTPJumpParams = self.isCallEnabled(["liftingheight","heightlimit"])
        self.GetCPParams = self.isCallEnabled(["cpvelocity","cpacceleration"])
        self.GetARCParams =
self.isCallEnabled(["arcxyzvelocity","arcrvelocity","arcxyzacceleration","arcracceleration"])
```

```
        self.GetAngleSensorStaticError = self.isCallEnabled(["anglestaticerrrear","anglestaticerrfront"])
        self.GetAngleSensorCoef = self.isCallEnabled(["anglecoefrear","anglecoeffront"])
        self.GetJOGLParams = self.isCallEnabled(["slidingrailjogvelocity","slidingrailjogacceleration"])
        self.GetPTPLParams = self.isCallEnabled(["slidingrailptpvelocity","slidingrailptpacceleration"])

    def fetch(self):
        if self.isEnabled("devicetime"):
            Dobot.deviceTime.labels(device_id=self.id, device_type=self.type,
station=self.host).set(dTypeX.GetDeviceTime(self.api)[0])

        if self.isEnabled("queueindex"):
            Dobot.queueIndex.labels(device_id=self.id, device_type=self.type,
station=self.host).set(dTypeX.GetQueuedCmdCurrentIndex(self.api)[0])

        if self.GetPose:
            pose = dTypeX.GetPose(self.api)
            if self.isEnabled("posex"):
                Dobot.poseX.labels(device_id=self.id, device_type=self.type, station=self.host).set(pose[0])

            if self.isEnabled("posey"):
                Dobot.poseY.labels(device_id=self.id, device_type=self.type, station=self.host).set(pose[1])

            if self.isEnabled("posez"):
                Dobot.poseZ.labels(device_id=self.id, device_type=self.type, station=self.host).set(pose[2])

            if self.isEnabled("poser"):
                Dobot.poseR.labels(device_id=self.id, device_type=self.type, station=self.host).set(pose[3])

            if self.isEnabled("anglebase"):
                Dobot.angleBase.labels(device_id=self.id, device_type=self.type, station=self.host).set(pose[4])

            if self.isEnabled("anglereararm"):
                Dobot.angleRearArm.labels(device_id=self.id, device_type=self.type,
station=self.host).set(pose[5])

            if self.isEnabled("angleforearm"):
                Dobot.angleForearm.labels(device_id=self.id, device_type=self.type, station=self.host).set(pose[6])
            if self.isEnabled("angleendeffector"):
                Dobot.angleEndEffector.labels(device_id=self.id, device_type=self.type,
station=self.host).set(pose[7])

        if self.isEnabled("alarmsstate"):
            alarmsList = dTypeX.GetAlarmsStateX(self.api)
            if len(alarmsList) == 0:
                Dobot.alarmsState.labels(device_id=self.id, device_type=self.type, station=self.host).state(" ")
            else:
                for a in alarmsList:
                    Dobot.alarmsState.labels(device_id=self.id, device_type=self.type, station=self.host).state(a)

        if self.GetHomeParams:
            home = dTypeX.GetHOMEParams(self.api)
            if self.isEnabled("homex"):
                Dobot.homeX.labels(device_id=self.id, device_type=self.type, station=self.host).set(home[0])

            if self.isEnabled("homey"):
                Dobot.homeY.labels(device_id=self.id, device_type=self.type, station=self.host).set(home[1])

            if self.isEnabled("homez"):
                Dobot.homeZ.labels(device_id=self.id, device_type=self.type, station=self.host).set(home[2])

            if self.isEnabled("homer"):
                Dobot.homeR.labels(device_id=self.id, device_type=self.type, station=self.host).set(home[3])

        if self.GetEndEffectorParams:
            endEffector = dTypeX.GetEndEffectorParams(self.api)
            if self.isEnabled("endeffectorx"):
                Dobot.endEffectorX.labels(device_id=self.id, device_type=self.type,
station=self.host).set(endEffector[0])

            if self.isEnabled("endeffectory"):
                Dobot.endEffectorY.labels(device_id=self.id, device_type=self.type,
station=self.host).set(endEffector[1])
```

```
            if self.isEnabled("endeffectorz"):
                Dobot.endEffectorZ.labels(device_id=self.id, device_type=self.type,
station=self.host).set(endEffector[2])

        if self.isEnabled("laserstatus"):
            if bool(dTypeX.GetEndEffectorLaser(self.api)[0]):
                Dobot.laserStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("enabled")
            else:
                Dobot.laserStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("disabled")

        if self.isEnabled("suctioncupstatus"):
            if bool(dTypeX.GetEndEffectorSuctionCup(self.api)[0]):
                Dobot.suctionCupStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("enabled")
            else:
                Dobot.suctionCupStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("disabled")

        if self.isEnabled("gripperstatus"):
            if bool(dTypeX.GetEndEffectorGripper(self.api)[0]):
                Dobot.gripperStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("enabled")
            else:
                Dobot.gripperStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("disabled")

        if self.GetJOGGointParams:
            jogJoints = dTypeX.GetJOGJointParams(self.api)
            if self.isEnabled("jogbasevelocity"):
                Dobot.jogBaseVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[0])

            if self.isEnabled("jogreararmvelocity"):
                Dobot.jogRearArmVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[1])
```

```
            if self.isEnabled("jogforearmvelocity"):
                Dobot.jogForearmVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[2])

            if self.isEnabled("jogendeffectorvelocity"):
                Dobot.jogEndEffectorVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[3])

            if self.isEnabled("jogbaseacceleration"):
                Dobot.jogBaseAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[4])

            if self.isEnabled("jogreararmacceleration"):
                Dobot.jogRearArmAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[5])

            if self.isEnabled("jogforearmacceleration"):
                Dobot.jogForearmAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[6])

            if self.isEnabled("jogendeffectoracceleration"):
                Dobot.jogEndEffectorAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogJoints[7])

        if self.GetJOGCoordinateParams:
            jogCoords = dTypeX.GetJOGCoordinateParams(self.api)
            if self.isEnabled("jogaxisxvelocity"):
                Dobot.jogAxisXVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[0])

            if self.isEnabled("jogaxisyvelocity"):
                Dobot.jogAxisYVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[1])

            if self.isEnabled("jogaxiszvelocity"):
                Dobot.jogAxisZVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[2])

            if self.isEnabled("jogaxisrvelocity"):
                Dobot.jogAxisRVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[3])

            if self.isEnabled("jogaxisxacceleration"):
                Dobot.jogAxisXAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[4])

            if self.isEnabled("jogaxisyacceleration"):
                Dobot.jogAxisYAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[5])

            if self.isEnabled("jogaxiszacceleration"):
                Dobot.jogAxisZAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[6])

            if self.isEnabled("jogaxisracceleration"):
                Dobot.jogAxisRAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCoords[7])

        if self.GetJOGCommonParams:
            jogCommon = dTypeX.GetJOGCommonParams(self.api)
            if self.isEnabled("jogvelocityratio"):
                Dobot.jogVelocityRatio.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCommon[0])

            if self.isEnabled("jogaccelerationratio"):
                Dobot.jogAccelerationRatio.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogCommon[1])

        if self.GetPTPJointParams:
            ptpJoints = dTypeX.GetPTPJointParams(self.api)
            if self.isEnabled("ptpbasevelocity"):
```

```
            Dobot.ptpBaseVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[0])

        if self.isEnabled("ptprearamvelocity"):
            Dobot.ptpRearArmVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[1])

        if self.isEnabled("ptpforearmvelocity"):
            Dobot.ptpForearmVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[2])

        if self.isEnabled("ptpendeffectorvelocity"):
            Dobot.ptpEndEffectorVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[3])

        if self.isEnabled("ptpbaseacceleration"):
            Dobot.ptpBaseAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[4])

        if self.isEnabled("ptprearamacceleration"):
            Dobot.ptpRearArmAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[5])

        if self.isEnabled("ptpforearmacceleration"):
            Dobot.ptpForearmAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[6])

        if self.isEnabled("ptpendeffectoracceleration"):
            Dobot.ptpEndEffectorAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJoints[7])

    if self.GetPTPCoordinateParams:
        ptpCoords = dTypeX.GetPTPCoordinateParams(self.api)
        if self.isEnabled("ptpaxisxyzvelocity"):
            Dobot.ptpAxisXYZVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpCoords[0])
```

```
            if self.isEnabled("ptpaxisrvelocity"):
                Dobot.ptpAxisRVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpCoords[1])

            if self.isEnabled("ptpaxisxyzacceleration"):
                Dobot.ptpAxisXYZAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpCoords[2])

            if self.isEnabled("ptpaxisracceleration"):
                Dobot.ptpAxisRAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpCoords[3])

        if self.GetPTPCommonParams:
            ptpCommon = dTypeX.GetPTPCommonParams(self.api)
            if self.isEnabled("ptpvelocityratio"):
                Dobot.ptpVelocityRatio.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpCommon[0])

            if self.isEnabled("ptpaccelerationratio"):
                Dobot.ptpAccelerationRatio.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpCommon[1])

        if self.GetPTPJumpParams:
            ptpJump = dTypeX.GetPTPJumpParams(self.api)
            if self.isEnabled("liftingheight"):
                Dobot.liftingHeight.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJump[0])

            if self.isEnabled("heightlimit"):
                Dobot.heightLimit.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpJump[1])

        if self.GetCPParams:
            cp = dTypeX.GetCPParams(self.api)
            if self.isEnabled("cpvelocity"):
                Dobot.cpVelocity.labels(device_id=self.id, device_type=self.type, station=self.host).set(cp[0])
            if self.isEnabled("cpacceleration"):
                Dobot.cpAcceleration.labels(device_id=self.id, device_type=self.type, station=self.host).set(cp[1])

        if self.GetARCParams:
            arc = dTypeX.GetARCParams(self.api)
            if self.isEnabled("arcxyzvelocity"):
                Dobot.arcXYZVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(arc[0])

            if self.isEnabled("arcrvelocity"):
                Dobot.arcRVelocity.labels(device_id=self.id, device_type=self.type, station=self.host).set(arc[1])

            if self.isEnabled("arcxyzacceleration"):
                Dobot.arcXYZAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(arc[2])

            if self.isEnabled("arcracceleration"):
                Dobot.arcRAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(arc[3])

        if self.GetAngleSensorStaticError:
            angleStaticErr = dTypeX.GetAngleSensorStaticError(self.api)
            if self.isEnabled("anglestaticerrrear"):
                Dobot.angleStaticErrRear.labels(device_id=self.id, device_type=self.type,
station=self.host).set(angleStaticErr[0])

            if self.isEnabled("anglestaticerrfront"):
                Dobot.angleStaticErrFront.labels(device_id=self.id, device_type=self.type,
station=self.host).set(angleStaticErr[1])

        if self.GetAngleSensorCoef:
            angleCoef = dTypeX.GetAngleSensorCoef(self.api)
            if self.isEnabled("anglecoefrear"):
                Dobot.angleCoefRear.labels(device_id=self.id, device_type=self.type,
station=self.host).set(angleCoef[0])

            if self.isEnabled("anglecoeffront"):
                Dobot.angleCoefFront.labels(device_id=self.id, device_type=self.type,
station=self.host).set(angleCoef[1])

        if self.isEnabled("slidingrailstatus"):
            if bool(dTypeX.GetDeviceWithL(self.api)[0]):
                Dobot.slidingRailStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("enabled")
            else:
                Dobot.slidingRailStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("disabled")

        if self.isEnabled("slidingrailpose"):
            Dobot.slidingRailPose.labels(device_id=self.id, device_type=self.type,
station=self.host).set(dTypeX.GetPoseL(self.api)[0])

        if self.GetJOGLParams:
            jogRail = dTypeX.GetJOGLParams(self.api)
            if self.isEnabled("slidingrailjogvelocity"):
                Dobot.slidingRailJogVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogRail[0])

            if self.isEnabled("slidingrailjogacceleration"):
                Dobot.slidingRailJogAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(jogRail[1])

        if self.GetPTPLParams:
            ptpRail = dTypeX.GetPTPLParams(self.api)
            if self.isEnabled("slidingrailptpvelocity"):
                Dobot.slidingRailPtpVelocity.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpRail[0])

            if self.isEnabled("slidingrailptpacceleration"):
                Dobot.slidingRailPtpAcceleration.labels(device_id=self.id, device_type=self.type,
station=self.host).set(ptpRail[1])

        if self.isEnabled("wifimodulestatus"):
            if bool(dTypeX.GetWIFIConfigMode(self.api)[0]):
                Dobot.wifiModuleStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("enabled")
            else:
                Dobot.wifiModuleStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("disabled")

        if self.isEnabled("wificonnectionstatus"):
            if bool(dTypeX.GetWIFIConnectStatus(self.api)[0]):
                Dobot.wifiConnectionStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("enabled")
            else:
                Dobot.wifiConnectionStatus.labels(device_id=self.id, device_type=self.type,
station=self.host).state("disabled")

    def disconnect(self):
        dTypeX.DisconnectDobotX(self.api)

class Jevois(Device):
    options = {"objects":"","objectidentified":True,"objectlocation":True,"objectsize":False}

    objectLocationX = Gauge("object_location_x", "Identified object's x position",
["device_id","device_type","station"])
    objectLocationY = Gauge("object_location_y", "Identified object's y position",
["device_id","device_type","station"])
    objectLocationZ = Gauge("object_location_z", "Identified object's Z position",
["device_id","device_type","station"])
    objectSize = Gauge("object_size","Identified object's size", ["device_id","device_type","station"])

    def connect(self):
        try:
            self.serial = serial.Serial(self.port, 115200, timeout=0)
            self.__initialize()
        except Exception as e:
            raise Exception(str(e))

    def __initialize(self):
        if self.isEnabled("objectidentified"):
            if self.section["objects"] is not None:
                self.objects = [" "]
                # remove extension part and add to the objects list
                for obj in self.section["objects"].split():
                    self.objects.append(obj.split(".")[0])

                self.objectIdentified = Enum("object_id_"+self.port, "Object Identified",
["device_id","device_type","station"], states=self.objects)
            else:
                raise Exception("The \"objects\" list is necessary for monitoring identified objects")

    def fetch(self):
        line = self.serial.readline().rstrip().decode()
        tok = line.split()

        # in case of no identified object (empty message) or malformed line (as a message with Normal serstyle
has 6 fields) skip fetching
        if len(tok) < 6:
            if self.isEnabled("objectidentified"):
                self.objectIdentified.labels(device_id=self.id, device_type=self.type, station=self.host).state(" ")
            Jevois.objectLocationX.labels(device_id=self.id, device_type=self.type, station=self.host).set(0)
            Jevois.objectLocationY.labels(device_id=self.id, device_type=self.type, station=self.host).set(0)
            Jevois.objectLocationZ.labels(device_id=self.id, device_type=self.type, station=self.host).set(0)
            Jevois.objectSize.labels(device_id=self.id, device_type=self.type, station=self.host).set(0)
            self.serial.flushInput()
            return

        serstyle = tok[0][0]
        dimension = tok[0][1]

        # If the serstyle is not Normal (thus it is not supported by the module)
        if (serstyle != "N"): raise Exception("Unsupported serstyle (" + serstyle + ")")

        if dimension == "1" and len(tok) != 4: raise Exception("Malformed line (expected 4 fields but received "
+ str(len(tok)) + ")")
        if dimension == "2" and len(tok) != 6: raise Exception("Malformed line (expected 6 fields but received "
+ str(len(tok)) + ")")
```

```
    if dimension == "3" and len(tok) != 8: raise Exception("Malformed line (expected 8 fields but received "
+ str(len(tok)) + ")")

    if self.isEnabled("objectidentified"):
      if len(self.objects) > 1:
        obj = tok[1].split(".")[0]
        if obj in self.objects:
          self.objectIdentified.labels(device_id=self.id, device_type=self.type, station=self.host).state(obj)
        else:
          self.objectIdentified.labels(device_id=self.id, device_type=self.type, station=self.host).state(" ")
      else:
        raise Exception("The \"objects\" list exists but is empty")

    if self.isEnabled("objectlocation"):
      Jevois.objectLocationX.labels(device_id=self.id, device_type=self.type,
station=self.host).set(float(tok[2]))

      if int(dimension) > 1:
        Jevois.objectLocationY.labels(device_id=self.id, device_type=self.type,
station=self.host).set(float(tok[3]))

      if int(dimension) == 3:
        Jevois.objectLocationZ.labels(device_id=self.id, device_type=self.type,
station=self.host).set(float(tok[4]))

    if self.isEnabled("objectsize"):
      if dimension == "1":
        Jevois.objectSize.labels(device_id=self.id, device_type=self.type,
station=self.host).set(float(tok[3]))
      elif dimension == "2":
        Jevois.objectSize.labels(device_id=self.id, device_type=self.type,
station=self.host).set(abs(float(tok[4])*float(tok[5])))
      elif dimension == "3":
        Jevois.objectSize.labels(device_id=self.id, device_type=self.type,
station=self.host).set(abs(float(tok[5])*float(tok[6])*float(tok[7])))

    self.serial.flushInput()
```

```
def disconnect(self):
    self.serial.close()
```

Appendix D

Source code of device modules testing utility (test.py)

```
'''
A small manageable testing utility for the monitoring agent (agent.py) is the `test.py` script
which includes a number of functions respective to different functional (f) and performance (p) tests.
Each functional test represents a function of the monitoring agent (agent.py).
Each test returns true in successful completion and false otherwise.
Performance tests produce results/statistics to standard output that can be further analyzed.
The naming convention for better organization and use of the test functions
is as follows: typeOfTest_moduleName_description
e.g.    For a performance test regarding the Jevois module => p_Jevois_DescriptionOfTest()
        For a functional test regarding the Dobot module => f_Dobot_DescriptionOfTest()
'''
import sys, os
import threading
import time
import serial
import configparser
import runtime.DobotDllTypeX as dTypeX

### Performance Tests ###
'''
Description: Helper function to measureFetchingOverheadDobot()
Execute a dobot function and return the execution time in ms
Parameters:
    func: (dTypeX) api call to be measured
    arg: dobot api object
'''
def _execute(func, arg):
    start = time.time()
    func(arg)
    stop = time.time()
    return (stop - start) * 1000
```

```
'''
Description: Measure fetching times (in ms) for each attribute through multiple iterations
Parameters:
    port: dobot device port (e.g. "COM4" / "192.168.43.4")
    n: number of iterations
'''
def p_Dobot_FetchingOverhead(port, n):
    dobot, state = dTypeX.ConnectDobotX(port)
    if state[0] != dTypeX.DobotConnect.DobotConnect_NoError:
        print("Couldn't connect Dobot at port " + str(port))
        return False

    if n < 1:
        print("Iterations number must be a positive integer.")
        return False

    iterations = n
    attributes =
{"GetDeviceSN":[],"GetDeviceName":[],"GetDeviceVersion":[],"GetWIFISSID":[],"GetWIFIPassword":[],"
GetWIFIIPAddress":[],

"GetWIFINetmask":[],"GetWIFIGateway":[],"GetWIFIDNS":[],"GetDeviceTime":[],"GetQueuedCmdCurren
tIndex":[],
    "GetPose":[],"GetAlarmsStateX":[],"GetHOMEParams":[],"GetEndEffectorParams":[],

"GetEndEffectorLaser":[],"GetEndEffectorSuctionCup":[],"GetEndEffectorGripper":[],"GetJOGJointParams"
:[],

"GetJOGCoordinateParams":[],"GetJOGCommonParams":[],"GetPTPJointParams":[],"GetPTPCoordinatePar
ams":[],

"GetPTPCommonParams":[],"GetPTPJumpParams":[],"GetCPParams":[],"GetARCParams":[],"GetAngleSen
sorStaticError":[],
    "GetAngleSensorCoef":[],"GetDeviceWithL":[],"GetPoseL":[],"GetJOGLParams":[],"GetPTPLParams":[],
    "GetWIFIConfigMode":[],"GetWIFIConnectStatus":[]}

    for i in range(iterations):
```

```
    attributes["GetDeviceSN"].append(_execute(dTypeX.GetDeviceSN, dobot))
    attributes["GetDeviceName"].append(_execute(dTypeX.GetDeviceName, dobot))
    attributes["GetDeviceVersion"].append(_execute(dTypeX.GetDeviceVersion, dobot))
    attributes["GetWIFISSID"].append(_execute(dTypeX.GetWIFISSID, dobot))
    attributes["GetWIFIPassword"].append(_execute(dTypeX.GetWIFIPassword, dobot))
    attributes["GetWIFIIPAddress"].append(_execute(dTypeX.GetWIFIIPAddress, dobot))
    attributes["GetWIFINetmask"].append(_execute(dTypeX.GetWIFINetmask, dobot))
    attributes["GetWIFIGateway"].append(_execute(dTypeX.GetWIFIGateway, dobot))
    attributes["GetWIFIDNS"].append(_execute(dTypeX.GetWIFIDNS, dobot))
    attributes["GetDeviceTime"].append(_execute(dTypeX.GetDeviceTime, dobot))
    attributes["GetQueuedCmdCurrentIndex"].append(_execute(dTypeX.GetQueuedCmdCurrentIndex,
dobot))
    attributes["GetPose"].append(_execute(dTypeX.GetPose, dobot))
    attributes["GetAlarmsStateX"].append(_execute(dTypeX.GetAlarmsStateX, dobot))
    attributes["GetHOMEParams"].append(_execute(dTypeX.GetHOMEParams, dobot))
    attributes["GetEndEffectorParams"].append(_execute(dTypeX.GetEndEffectorParams, dobot))
    attributes["GetEndEffectorLaser"].append(_execute(dTypeX.GetEndEffectorLaser, dobot))
    attributes["GetEndEffectorSuctionCup"].append(_execute(dTypeX.GetEndEffectorSuctionCup, dobot))
    attributes["GetEndEffectorGripper"].append(_execute(dTypeX.GetEndEffectorGripper, dobot))
    attributes["GetJOGJointParams"].append(_execute(dTypeX.GetJOGJointParams, dobot))
    attributes["GetJOGCoordinateParams"].append(_execute(dTypeX.GetJOGCoordinateParams, dobot))
    attributes["GetJOGCommonParams"].append(_execute(dTypeX.GetJOGCommonParams, dobot))
    attributes["GetPTPJointParams"].append(_execute(dTypeX.GetPTPJointParams, dobot))
    attributes["GetPTPCoordinateParams"].append(_execute(dTypeX.GetPTPCoordinateParams, dobot))
    attributes["GetPTPCommonParams"].append(_execute(dTypeX.GetPTPCommonParams, dobot))
    attributes["GetPTPJumpParams"].append(_execute(dTypeX.GetPTPJumpParams, dobot))
    attributes["GetCPParams"].append(_execute(dTypeX.GetCPParams, dobot))
    attributes["GetARCParams"].append(_execute(dTypeX.GetARCParams, dobot))
    attributes["GetAngleSensorStaticError"].append(_execute(dTypeX.GetAngleSensorStaticError, dobot))
    attributes["GetAngleSensorCoef"].append(_execute(dTypeX.GetAngleSensorCoef, dobot))
    attributes["GetDeviceWithL"].append(_execute(dTypeX.GetDeviceWithL, dobot))
    attributes["GetPoseL"].append(_execute(dTypeX.GetPoseL, dobot))
    attributes["GetJOGLParams"].append(_execute(dTypeX.GetJOGLParams, dobot))
    attributes["GetPTPLParams"].append(_execute(dTypeX.GetPTPLParams, dobot))
    attributes["GetWIFIConfigMode"].append(_execute(dTypeX.GetWIFIConfigMode, dobot))
    attributes["GetWIFIConnectStatus"].append(_execute(dTypeX.GetWIFIConnectStatus, dobot))
print("\nResults for " + str(iterations) + " iterations (attribute_name, avg, min, max (in milliseconds))")
for attr in attributes:
    max = 0
    min = sys.maxsize
    sum = 0
    for i in range(iterations):
        ms = attributes[attr][i]
        sum += ms
        if ms > max:
            max = ms
        if ms < min:
            min = ms

    print(str(attr) + "," + str(round(sum/iterations)) + "," + str(round(min)) + "," + str(round(max)))

return True

'''
Description: Measure the fetching rate in which one can receive standardized messages through the serial port
Parameters:
port: jevois serial port
n: number of iterations
'''
def p_Jevois_FetchingRate(port, n):
try:
    ser = serial.Serial(port, 115200, timeout=0)
    line = ser.readline().rstrip()
    tok = line.split()
    print("Jevois Camera at port " + port + " connected succesfully!")
except Exception as e:
    print("Couldn't connect with Jevois Camera device at port " + port + " (" + str(e) + ")")
    return False

if n < 1:
    print("Iterations number must be a positive integer.")
    return False
iterations = n

max = 0
min = sys.maxsize
sum = 0
i = 0
while i < iterations:
    start = time.time()
    line = ser.readline().rstrip().decode()
    stop = time.time()
    if len(line) > 0:
        print(line)
        ms = (stop - start) * 1000
        sum += ms
        if ms > max:
            max = ms
        if ms < min:
            min = ms
        i += 1

print("\nResults for " + str(iterations) + " iterations (attribute_name, avg, min, max (in milliseconds))")
print(str(round(sum/iterations)) + "," + str(round(min)) + "," + str(round(max)))
return True

'''
Description: Measure the switching overhead of connecting to multiple dobot devices
through a 2 dobot connection paradigm using the default dTypeX.ConnectDobot() call
Parameters:
port1: dobot device port (e.g. "COM4" / "192.168.43.4")
port2: dobot device port (e.g. "COM4" / "192.168.43.4")
'''
def p_Dobot_SwitchOverhead(port1, port2):
# Run for 1 minute
stop = time.time() + 60
sum = 0
count = 0
api = dTypeX.load()
while (time.time() < stop):
    state = dTypeX.ConnectDobot(api, port1, 115200)[0]
    if state != dTypeX.DobotConnect.DobotConnect_NoError:
        print("Can't connect to the dobot. Aborting test.")
        return False
    start = time.time()
    dTypeX.DisconnectDobot(api)
    state = dTypeX.ConnectDobot(api, port2, 115200)[0]
    if state != dTypeX.DobotConnect.DobotConnect_NoError:
        print("Can't connect to the dobot. Aborting test.")
        return False
    end = time.time()
    sum += end - start
    count += 1
    print(f"Switching overhead is {end - start} \n")

print("Recorded %3d switches\n"% (count))
print("Total overhead: %5.2f seconds\n"% (sum))
print("Average Switch Overhead: %5.2f seconds\n"% (sum / count))
print("Percentage of time spend on overhead %5.2f%%\n"% (sum * 100 / 60))
return True

### Functional Tests ###
'''
Description: Test connecting to a JeVois camera device
Parameters:
port: jevois device port (e.g. "COM3")
'''
def f_Jevois_Connectivity(port):
try:
    ser = serial.Serial(port, 115200, timeout=0.25)
    print("Jevois Camera at port " + port + " connected succesfully!")
except Exception as e:
    print("Couldn't connect with Jevois Camera device at port " + port + " (" + str(e) + ")")
    return False

stop = time.time() + 30
while (time.time() < stop):
    line = ser.readline().rstrip().decode()
    tok = line.split()
    # Abort fetching if timeout or malformed line
    if len(tok) < 1: print("No data found"); continue
    if tok[0][0] != 'N': print("Unsupported serstyle"); continue
    if tok[0][1] == '1' and len(tok) != 4: continue
    elif tok[0][1] == '2' and len(tok) != 6: continue
    elif tok[0][1] == '3' and len(tok) != 8: continue
    print(line)

return True

'''
Description: Test the non-standard dTypeX.GetAlarmsStateX() function to fetch alarms
created as an alternative to the standard dTypeX.GetAlarmsState()
Parameters:
port: dobot device port (e.g. "COM4" / "192.168.43.4")
'''
def f_Dobot_Alarms(port):
dobot, state = dTypeX.ConnectDobotX(port)

if state[0] != dTypeX.DobotConnect.DobotConnect_NoError:
    return False

print(port + "\'s name: " + str(dTypeX.GetDeviceName(dobot)[0]))

stop = time.time() + 60
while (time.time() < stop):
    print(time.time + "Active alarms:")
    for a in dTypeX.GetAlarmsStateX(dobot):
        print(a)
    time.sleep(0.2)

dTypeX.DisconnectAll()
return True
```

```
'''
Description: Test the non-standard dTypeX.ConnectDobotX() function to connect to multiple
dobot devices in parallel (and diminish switching overhead), created as
an alternative to the standard dTypeX.ConnectDobot()
Parameters:
    portList: a list of strings indicating the multiple ports of dobot devices
    (e.g. ["192.168.43.4","192.168.43.5"])
'''
def f_Dobot_ParallelConnection(portList):
    dobotList = []
    for port in portList:
        dobot, state = dTypeX.ConnectDobotX(port)

        if state[0] == dTypeX.DobotConnect.DobotConnect_NoError:
            dobotList.append(dobot)
        else:
            return False

    print("\nConnected Dobots:")
    for dobot in dobotList:
        print("Device Name: " + str(dTypeX.GetDeviceName(dobot)[0]))
        print("Device Serial No: " + str(dTypeX.GetDeviceSN(dobot)[0]))
        print("Device Version: " + ".".join(list(map(str, dTypeX.GetDeviceVersion(dobot)))))
        print()

    dTypeX.DisconnectAll()
    return True

### Enable/Disable Tests ###
#p_Dobot_FetchingOverhead("192.168.43.4", 30)
#p_Jevois_FetchingRate("COM4", 30)
#p_Dobot_SwitchOverhead("192.168.43.4","192.168.43.5")
#f_Jevois_Connectivity("COM3")
#f_Dobot_Alarms("192.168.43.4")
#f_Dobot_ParallelConnection(["192.168.43.4","192.168.43.5"])
```

Appendix E

Example of device discovery/configuration file used by the monitoring agent (devices.conf)

```
[Dobot:192.168.43.4]
    Timeout = 500
    DeviceSN: on
    DeviceName: on
    DeviceVersion: on
    DeviceTime: off
    QueueIndex: off
    PoseX: on
    PoseY: on
    PoseZ: on
    PoseR: on
    AngleBase: on
    AngleRearArm: on
    AngleForearm: on
    AngleEndEffector: on
    AlarmsState: on
    HomeX: off
    HomeY: off
    HomeZ: off
    HomeR: off
    EndEffectorX: off
    EndEffectorY: off
    EndEffectorZ: off
    LaserStatus: off
    SuctionCupStatus: off
    GripperStatus: off

    # JOG mode related options
    JogBaseVelocity: off
    JogRearArmVelocity: off
    JogForearmVelocity: off
    JogEndEffectorVelocity: off
```

```
JogBaseAcceleration: off
JogRearArmAcceleration: off
JogForearmAcceleration: off
JogEndEffectorAcceleration: off
JogAxisXVelocity: off
JogAxisYVelocity: off
JogAxisZVelocity: off
JogAxisRVelocity: off
JogAxisXAcceleration: off
JogAxisYAcceleration: off
JogAxisZAcceleration: off
JogAxisRAcceleration: off
JogVelocityRatio: off
JogAccelerationRatio: off

# PTP mode related options
PtpBaseVelocity: off
PtpRearArmVelocity: off
PtpForearmVelocity: off
PtpEndEffectorVelocity: off
PtpBaseAcceleration: off
PtpRearArmAcceleration: off
PtpForearmAcceleration: off
PtpEndEffectorAcceleration: off
PtpAxisXYZVelocity: off
PtpAxisRVelocity: off
PtpAxisXYZAcceleration: off
PtpAxisRAcceleration: off
PtpVelocityRatio: off
PtpAccelerationRatio: off
LiftingHeight: off
HeightLimit: off

# CP mode related options
CpVelocity: off
CpAcceleration: off
```

```
    # ARC mode related options
    ArcXYZVelocity: off
    ArcRVelocity: off
    ArcXYZAcceleration: off
    ArcRAcceleration: off

    # Angle sensor options
    AngleStaticErrRear: off
    AngleStaticErrFront: off
    AngleCoefRear: off
    AngleCoefFront: off

    # Sliding rail options
    SlidingRailStatus: off
    SlidingRailPose: off
    SlidingRailJogVelocity: off
    SlidingRailJogAcceleration: off
    SlidingRailPtpVelocity: off
    SlidingRailPtpAcceleration: off

    # Wifi related options
    WifiModuleStatus: off
    WifiConnectionStatus: off
    WifiSSID: off
    WifiPassword: off
    WifiIPAddress: off
    WifiNetmask: off
    WifiGateway: off
    WifiDNS: off

[Dobot:192.168.43.5]
    DeviceSN: on
    DeviceName: on
    DeviceVersion: on
    DeviceTime: off
    QueueIndex: off
    PoseX: on
    PoseY: on
    PoseZ: on
    PoseR: on
    AngleBase: on
    AngleRearArm: on
    AngleForearm: on
    AngleEndEffector: on
    AlarmsState: on
    HomeX: off
    HomeY: off
    HomeZ: off
    HomeR: off
    EndEffectorX: off
    EndEffectorY: off
    EndEffectorZ: off
    LaserStatus: off
    SuctionCupStatus: off
    GripperStatus: off

[Jevois:COM3]
    objects = example pen
    ObjectIdentified: on
    ObjectLocation: on
    ObjectSize: off
```