muPlantPython

Release 0.1

Lennart Schink

May 25, 2023

TABLE OF CONTENTS

This Software is part of muPlant Project of University of Kassel. It implements basic functions for WareHouse Management.

CHAPTER

ONE

COMMUNICATION STANDARDS

- TCP/IP for communication with ABB Robot
- OPC UA with other muPlant stations
- RFID to communicate with turtle bots
- uEye Camera with openCV and arUco markers for automated storage detection

CHAPTER

TWO

LEARNINGS FROM EARLIER STUDIES

Sebastian Hübler has published hie practical studies in 2019. He evaluated methods to detect cups in muPlant storage by using two different cameras. Regarding the detection with arUco markers he made some helpful analysis:

- low resolution leads to better detection results
- minimal/ maximal distance uEye tp marker: 17mm/ 745mm
- ambient light has significant influence
- auto focus maybe a big issue
- detection best works while robotic arm is without movement

2.1 Further thoughts:

- if ambient light and other surcumstances are not good:
- reduce image size to criticl area only
- calibrate location of camera to maximize image size reduction
- better detection with custom filter which smoothes and increases contrast?
- other camera which has no autofocus

CHAPTER

THREE

INDICES AND TABLES

3.1 Getting started

- install Python 3.11
- install numpy
- install asynchua
- install pyside6
- install websocket
- install opencv

3.2 Modules

3.2.1 Modules and Scripts

This list contains all created Modules and scripts created for this software.

This is the entrancee file of Python-Implementation of muPlant Warehouse Manager. Author: L.Schink Date: 11.05.2023

This Python File implements the logic to recognize arUco markers. class VideoThread inherits from QThread class. So image capture and image processing code is in seperated thread. Processed images are provided to qml by using class videoPlayer which inherits from QQuickImageProvider.

class src.cameraApplication.cameraProcessing.**VideoPlayer**

requestImage(*id*, *size*, *requestedSize*)

This function overrides requestImage from inherited class. :param id: necessary identifier to switch between images. Can be any value. Implemented as boolean value which is toggled everytime when imageChanged is emitted form a JavaScript - function in CameraApplicationMain.qml :param size: :param requestedSize: :return: returns QImage object in RGBA color format

start()

Overrides start method of inherited class QQuickImageProvider. It is a Slot and called from QML Button of CameraAppMain.qml :return: this method returns nothing.

stop()

Overrides stop method of inherited class QQuickImageProvider. It is a Slot and called from QML Button of CameraAppMain.qml :return: this method returns nothing

toggleDetection()

Toggles detection field of VideoThread object. Enables / disables feature detection in VideoThread's run method. It is a Slot and called from QML Button of CameraAppMain.qml :return: This method returns nothing

updateImage(*frame*)

Implements connection between VideoThread and VideoPlayer. If VideoThread emits a new image this Slot is called. stores emitted image in self.image and emits image to QQmlEngine :param frame: QImage which is emitted from run-method in VideoThread object. :return: this method returns nothing but emits signal to QQmlEngine

class src.cameraApplication.cameraProcessing.**VideoThread**(*parent=None*)

capture

initializes the first camera device.

detect()

enables/disables detection in run-method

detecting

enables/disables feature detection

faceCascade

initialize haar cascade face detection.. just that there is some image processing

frameChanged

Signal which is emitted when a new image is ready for QQuickImageProvider

quit()

Necessary Implementation of inherited class to quit existing thread.

run()

This Method reads the camera sensor and performs necessary image processing. Converts processed image to Qt's QImage class and emits Signal with QImage

running

run variable for while loop in run() function

start()

Necessary Implementation of inherited class to quit existing thread.

class src.controller.EventlogController.**EventlogController**

class src.controller.InventoryController.**InventoryController**(*model:* InventoryModel = *None*, *parent=None*, *eventcontroller:* EventlogController = *None*, *productlist:* ProductListModel = *None*)

changeStorage(*storage*, *slot*, *cupID*, *productID*)

Takes Data from Override Storage Dialog from Storage.qml Decodes Storage ID 'L1' to L'18' in row / col and checks for ValueErrors. changes InventoryModel Data depending on entries.

loadStorage(*storage: str*, *slot: str*)

Takes Data from Override Storage Dialog from Storage.qml Decodes Storage ID 'L1' to L'18' in row / col and checks for ValueErrors. returns productslot, cup ID and productListindex.

class src.controller.websocketController.**WebsocketController**(*controller: EventlogController*, *parent=None*)

class src.model.InventoryModel.**InventoryModel**(*storageData*, *parent=None*)

columnCount(*self*, *parent: PySide6.QtCore.QModelIndex | PySide6.QtCore.QPersistentModelIndex = Invalid(PySide6.QtCore.QModelIndex)*) → int

data(*self*, *index: PySide6.QtCore.QModelIndex | PySide6.QtCore.QPersistentModelIndex*, *role: int = Instance(Qt.DisplayRole)*) → Any

roleNames(*self*) → Dict[int, PySide6.QtCore.QByteArray]

rowCount(*self*, *parent: PySide6.QtCore.QModelIndex | PySide6.QtCore.QPersistentModelIndex = Invalid(PySide6.QtCore.QModelIndex)*) → int

setData(*self*, *index: PySide6.QtCore.QModelIndex | PySide6.QtCore.QPersistentModelIndex*, *value: Any*, *role: int = Instance(Qt.EditRole)*) → bool

class src.model.ProductListModel.**ProductListModel**(*products*, *parent=None*)

data(*index*, *role*)

Returns an appropriate value for the requested data. If the view requests an invalid index, an invalid variant is returned. Any valid index that corresponds to a string in the list causes that string to be returned :param index: :param role: :return:

headerData(*section*, *orientation*, *role=ItemDataRole.DisplayRole*)

Returns the appropriate header string depending on the orientation of the header and the section. If anything other than the display role is requested, we return an invalid variant.

roleNames(*self*) → Dict[int, PySide6.QtCore.QByteArray]

rowCount(*self*, *parent: PySide6.QtCore.QModelIndex | PySide6.QtCore.QPersistentModelIndex = Invalid(PySide6.QtCore.QModelIndex)*) → int

class src.model.ProductSummaryListModel.**InventoryFilterProxyModel**(*model*, *parent=None*)

filterAcceptsRow(*self*, *source_row: int*, *source_parent: PySide6.QtCore.QModelIndex | PySide6.QtCore.QPersistentModelIndex*) → bool

class src.model.ProductSummaryListModel.**ProductSummaryListModel**(*products*, *parent=None*)

data(*index*, *role*)

Returns an appropriate value for the requested data. If the view requests an invalid index, an invalid variant is returned. Any valid index that corresponds to a string in the list causes that string to be returned. :param index: :param role: :return:

headerData(*section*, *orientation*, *role=ItemDataRole.DisplayRole*)

Returns the appropriate header string depending on the orientation of the header and the section. If anything other than the display role is requested, we return an invalid variant :param section: :param orientation: :param role: :return:

roleNames(*self*) → Dict[int, PySide6.QtCore.QByteArray]

rowCount(*self*, *parent: PySide6.QtCore.QModelIndex | PySide6.QtCore.QPersistentModelIndex = Invalid(PySide6.QtCore.QModelIndex)*) → int

3.3 Contributions

- Qt Project

PYTHON MODULE INDEX

m

s