Solar Irradiance Microforecasting
Senior Project Spring 2018 - Fall 2018
California Polytechnic State University, San Luis Obispo
Under the direction of Professor William L. Ahlgren

Sara N. Kipps       Austin J. Fite

December 2018
Author Information

Austin J. Fite, Computer Engineering Undergraduate
Up to date contact information can be found on ajfite.com

Sara N. Kipps, Electrical Engineering Undergraduate

Acknowledgements

Special thanks to the University of New Mexico and Microgrid Labs for providing the equipment that made this project possible.
## Contents

1 Introduction .................................................. 3

2 Background .................................................... 3
   2.1 Theory .................................................. 3
   2.2 Hardware ............................................... 4

3 Control Interface ............................................ 6
   3.1 Summary of Installation and Use ......................... 8

4 Data Collection ............................................... 8

5 Image Processing .............................................. 9

6 Conclusion .................................................... 11
   6.1 Summary of Progress ................................... 11
   6.2 Moving Forward ........................................ 11
      6.2.1 Image Processing .................................. 11
      6.2.2 Control Interface .................................. 12

Appendices ..................................................... 13

A Works Cited .................................................. 13

B Code .......................................................... 13
   B.1 Control Interface ....................................... 13
      B.1.1 License ............................................. 13
      B.1.2 setup.py ........................................... 14
      B.1.3 main.py .............................................. 15
      B.1.4 guientry.py ......................................... 20
      B.1.5 filehandlers.py ...................................... 21
      B.1.6 config.py ........................................... 22
      B.1.7 about.py ............................................ 27
      B.1.8 sudo.py .............................................. 28
   B.2 Image Processing ........................................ 28
      B.2.1 SIMF_institch_test.m ................................ 28
      B.2.2 SIMF_suntracking.m .................................. 31
1 Introduction

In late winter quarter 2018 the University of New Mexico and Microgrid Labs provided the Cal Poly Electrical Engineering department with their prototype Solar Irradiance Microforecasting (SIMF) hardware. This hardware was provided to the University under the lone condition that data collected with it would be returned to Microgrid Labs to train their weather model. The goal of this senior project was to take the provided hardware and software and make it usable for students in the Cal Poly Electrical and Computer Engineering departments to hack and expand. The end goal with the hardware is near term cloud and weather prediction primarily for solar installations but the hardware is versatile and could be re-purposed easily for other uses. The following report details our teams efforts towards this goal.

2 Background

2.1 Theory

A method of forecasting the level of solar energy is currently patented by the University of New Mexico[1]. The purpose of the forecasting software is to make power switching more effective, and extend the battery life of the system it is integrated into.

A 250 kWh subset of the battery system, capable of delivering up to 500 kW of power, is used to offset cloud-driven variability, by delivering power when clouds suddenly occlude the sun, and by absorbing power when the sun re-emerges. The magnitude of the power delivered or absorbed by the batteries is based on the difference between the instantaneous power produced by the PV farm, and an underlying ‘smooth’ power. [2]
Figure 1: Solar irradiance signal sampled at 1-second interval (red), and smoothed with a trailing window (green) and with a centered window (blue) using a 4-minute window (top plot) and a 10-minute window (bottom plot). [2]

2.2 Hardware

The capture station hardware is designed using off the shelf affordable com-
Figure 3: Shot from inside Building 20 showing the typical data collection location, note the USB extender umbilical.

Figure 4: A close up of the camera dome as it is capturing data.
ponent to allow for a low cost end product. The sensor array consists of 8 GroupGets PureThermal 1 USB FLIR cameras and a LI-COR LI-200R Pyranometer, see figure 2 for a block diagram. The pyranometer is interfaced to the system board using an Arduino to the analog to digital conversion, the Arduino converts the analog signal from the pyranometer to text printed out of its USB TTY. The FLIR cameras are attached to the computer via active USB extenders for additional range and a high power 7 port Anker USB hub, the high power USB hub is essential for powering the USB extenders. The PC is an off the shelf ASRock single board computer with a generic power supply. Both the sensor dome and PC are mounted in waterproof boxes for permanent outdoor mounting.

3 Control Interface

The initial control interface provided to the group by the University of New Mexico was a command line python utility called Lepton-Grabber\(^1\). While this interface is sufficient for data collection it is not intuitive, especially for the less tech literate people who may be interested in the system.

To alleviate this problem a graphical user interface (GUI) for the capture device was created. The GUI controller was designed to be cross platform

\(^1\)This utility is closed source, to request it contact Professor Ahlgren, Microgrid Labs, or AJ Fite
and easy to maintain, its written in Python using the QT and pyQT libraries for the graphical elements. Python was chosen to allow a Cal Poly Electrical Engineering (EE) student to hack and expand the controller in the future as EE students at Cal Poly are required\(^2\) to take CPE 101 which is taught in Python.

The controller is made up of two main screens, the capture screen (figure 6) and the configuration screen (figure 7). In order to decouple the capturing system and controller the controller functions by spawning the original lepton-grabber utility in a thread, using QT threading, and pipes its output to the capture window. This allows the capture logic to be updated by the University of New Mexico without affecting the operation of the controller. Code for the controller can be found in section A

\(^2\)As of publishing, reference the EE 2017-2019 catalog requirements
Figure 7: The settings screen of the SIMF Python GUI, with a dark QT theme applied

3.1 Summary of Installation and Use

The entire controller is packaged as a standard Python package available from the Python package repository PyPi under the name SimfPythonGUI and installs a gui entry point into the python PATH executable as simfgui from your favorite terminal program. When launching the program a default configuration file is written to ~/.simfgui.ini, all of the settings found in this file can be edited from the GUI settings menu. Be sure to locate the directory of the lepton-grabber script and set the path in the settings file, use an absolute path to allow you to launch the GUI from any directory. From there data capture can be started using the start capture button and stopped with the corresponding button. Other useful information can be found in the Readme file published on the PyPi package page or the git repositories.

4 Data Collection

Throughout Spring and Summer quarter 2018 ten rounds of data was captured. These rounds consisted of between one and five hours of data, captured in ten second intervals. This data is used for the image processing (see section 5) to design a model for cloud prediction. As part of the conditions for receiving the free capture hardware the data was passed along to the researchers at the University of New Mexico and is available for study upon request. The captured

3As of publishing, available at this link: https://pypi.org/project/SimfPythonGUI/
Figure 8: Output from a camera on the capture dome showing the sun silhouetting a tree

data is made up of a png and csv file for each camera and a text file. These files are outputted each ten second interval that elapses. The png and csv files generated by the camera are equivalent, the png file is friendly human readable image and the csv contains the raw data from each pixel of the thermal camera’s sensor. The text file is the instantaneous measurement of the Licor sensor in watts per meter squared, the solar irradiance.

5 Image Processing

Originally, the project was meant to use the Neural Network, Computer Vision, and Image Processing toolboxes on MatLab. However, the images produced by the current hardware were of too low a quality to use in this way. We were able to enhance them by converting the .csv files also collected by the apparatus into grayscale photographs.
The most significant result of this project in the realm of Image Processing was stitching most of the images together into a single image. The images had to be manually rotated and matched together, again due to the low quality of the images. This low quality also prevented the last two images from being matched into this larger image due to a lack of distinguishing features.
6 Conclusion

6.1 Summary of Progress

The project remains very much unfinished, our project however successfully laid the ground work for future students to develop on the hardware. Our efforts have been made public under open source licenses and are available to future students for use in their projects with the SIMF setup.

6.2 Moving Forward

There remains work to be done on image processing, and our group didn’t even touch on the electronic controls aspect of the system. However we never intended to complete the entire system as that is a task that is far greater than the allotted two quarters.

6.2.1 Image Processing

In order to complete this project the following things need to happen with Image Processing:

- Finish stitching images together, would likely be done more easily with better quality images
- Recognize both the sun and other objects in the stitched together picture
• Detect movement of the sun towards any of these objects in the stitched together picture

6.2.2 Control Interface

The control interface is in serviceable condition and ready for use with the hardware. Due to time constraints there are a few "TODO" comments in the code pointing out helpful extra features that could be implemented that are ripe for future student project work. The interface is meant as a starting point for future students to build off of and integrate with to expand the functionality of the system.
Appendices

A  Works Cited


B  Code

B.1  Control Interface

The Python code for the control interface is reproduced here for reference purposes. Web access to the code can be found at the following two repositories which also contain the UI files and other ancillary support files.

- git.nclf.net/SIMF/simf-python-gui
- github.com/Goldman60/simf-python-gui
- pypi.org/project/SimfPythonGUI/

B.1.1  License

The MIT License (MIT)

Copyright ©2018 A.J Fite <ajfite.com> and Sara Kipps

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

B.1.2 setup.py

#!/usr/bin/env python

import setuptools
import os

with open("README.md", "r") as fh:
    long_description = fh.read()

version_file = open(os.path.join('SimfPythonGUI', 'VERSION'))
version = version_file.read().strip()

setuptools.setup(
    name="SimfPythonGUI",
    version=version,
    author="A.J. Fite, Sara Kipps",
    author_email="me@ajfite.com, skipps@calpoly.edu",
    description="A GUI controller for the closed source SIMF camera capture utility",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://git.nclf.net/SIMF/simf-python-gui",
    packages=setuptools.find_packages(),
    package_data={'SimfPythonGUI': ["*.ui", 'VERSION']},
    entry_points={
        'gui_scripts': ['simfgui = SimfPythonGUI.guientry:main'],
    },
    data_files=[('', ['LICENSE.md'])],
    install_requires=['PyQt5', 'watchdog'],
    license="MIT",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
)
B.1.3 main.py

#!/usr/bin/env python
import sys
import webbrowser
from PyQt5 import uic
import pkg_resources
from PyQt5.QtCore import QProcess, QProcessEnvironment, Qt
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QMainWindow, QApplication, QGraphicsScene
from .filehandlers import ImageThread, LicorThread
from .sudo import PasswordWindow
from .about import AboutDialog
from .config import Config, ConfigEditor

class MainWindow(QMainWindow):
    simfProcess = QProcess()

    def __init__(self):
        super().__init__(flags=Qt.Window)

        ui_file = pkg_resources.resource_filename(__name__, "MainWindow.ui")
        uic.loadUi(ui_file, self)

        self.passprompt = None
        self.configdialog = None
        self.aboutdialog = None

        self.png_watcher = ImageThread()
        self.png_watcher.new_image.connect(self.update_image)

        self.licor_watcher = LicorThread()
        self.licor_watcher.new_licor.connect(self.update_licor)

        # Register Events
        self.startCapButton.clicked.connect(self.start_capture)
        self.stopCapButton.clicked.connect(self.stop_capture)
        self.actionSourceCode.triggered.connect(
            lambda:
                MainWindow.open_link("https://git.nclf.net/SIMF/simf-python-gui")
        )
        self.actionLicense.triggered.connect(
            lambda:
                MainWindow.open_link("https://git.nclf.net/SIMF/simf-python-gui/
""blob/master/LICENSE.md")
        )

    # Other methods and functions
self.actionExit.triggered.connect(self.close)
self.actionSettings.triggered.connect(self.show_settings)
self.actionAbout.triggered.connect(self.show_about)

# Process setup
self.simfProcess.readyRead.connect(self.console_write)
self.simfProcess.started.connect(self.process_started)
self.simfProcess.finished.connect(self.process_finished)

# Ready to show the UI!
self.show()

self.cameraCount.display("OFF")
self.solarIrradiance.display("OFF")

# Init the scenes for the graphics displays
self.sceneN = QGraphicsScene()
self.imgN.setScene(self.sceneN)
self.sceneNE = QGraphicsScene()
self.imgNE.setScene(self.sceneNE)
self.sceneE = QGraphicsScene()
self.imgE.setScene(self.sceneE)
self.sceneSE = QGraphicsScene()
self.imgSE.setScene(self.sceneSE)
self.sceneS = QGraphicsScene()
self.imgS.setScene(self.sceneS)
self.sceneSW = QGraphicsScene()
self.imgSW.setScene(self.sceneSW)
self.sceneW = QGraphicsScene()
self.imgW.setScene(self.sceneW)
self.sceneNW = QGraphicsScene()
self.imgNW.setScene(self.sceneNW)
self.sceneCenter = QGraphicsScene()
self.imgCenter.setScene(self.sceneCenter)
self.current = 0

# Updates the image and determines the capture progress
def update_image(self, path):
    pix = QPixmap(path)
    if self.current == 0:
        self.sceneNW.addPixmap(pix)
    elif self.current == 1:
        self.sceneN.addPixmap(pix)
    elif self.current == 2:
        self.sceneE.addPixmap(pix)
self.sceneNE.addPixmap(pix)
elif self.current == 3:
    self.sceneW.addPixmap(pix)
elif self.current == 4:
    self.sceneCenter.addPixmap(pix)
elif self.current == 5:
    self.sceneE.addPixmap(pix)
elif self.current == 6:
    self.sceneSW.addPixmap(pix)
elif self.current == 7:
    self.sceneS.addPixmap(pix)
elif self.current == 8:
    self.sceneSE.addPixmap(pix)

# TODO: Capture progress
self.current = (self.current + 1) % Config.dbg_lepton_set

# Opens a link
@staticmethod
def open_link(link):
    webbrowser.open(link)

# Triggered when window closes, I know it isn't PEP8 compliant but thats
# the way pyqt5 is
def closeEvent(self, QCloseEvent):
    self.simfProcess.kill()  # Cleanly kill the simfprocess
    self.close()

def show_about(self):
    self.aboutdialog = AboutDialog()

def show_settings(self):
    self.configdialog = ConfigEditor()

# Toggle all the enable/disable options
# status=true lepton-grabber process running
# status=false lepton-grabber process stopped
# sorry
def button_toggle(self, status):
    self.capProgress.setEnabled(status)
    self.cameraCount.setEnabled(status)
    self.solarIrradiance.setEnabled(status)
    self.startCapButton.setDisabled(status)
    self.stopCapButton.setEnabled(status)
    self.capProgressLabel.setEnabled(status)
    self.cameraCountLabel.setEnabled(status)
self.solarIrradianceLabel.setEnabled(status)
self.imgN.setEnabled(status)
self.imgNE.setEnabled(status)
self.imgE.setEnabled(status)
self.imgSE.setEnabled(status)
self.imgS.setEnabled(status)
self.imgSW.setEnabled(status)
self.imgW.setEnabled(status)
self.imgNW.setEnabled(status)
self.imgCenter.setEnabled(status)
self.actionSettings.setDisabled(status)

if status:
    self.cameraCount.display(Config.dbg_lepton_set)
    self.statusLabel.setText("Status: Capturing")
else:
    self.cameraCount.display("OFF")
    self.solarIrradiance.display("OFF")
    self.statusLabel.setText("Status: Stopped")

# Triggered when the QProcess that runs the lepton-grabber runs
def process_started(self):
    self.button_toggle(True)
    self.current = 0
    self.png_watcher.start()

# Triggered when the QProcess that runs the lepton-grabber dies for any
# reason
def process_finished(self):
    self.button_toggle(False)

    self.console_write()
    self.console_write_line("Capture Ended")

    self.png_watcher.terminate()

# Fired when the observer started in process_started
# detects a new image from the lepton grabbers
def update_images(self):
    self.console_write_line("New images!
")

def update_licor(self, licor):
    self.solarIrradiance.display(licor)
    self.current += 1

    self.capProgress.setValue(18)
# Allows the console to handle \n newlines
def console_write_line(self, output):
    # the QPlainTextEdit widget doesn't like newlines
    # So I use appendhtml and <br /> instead
    output = output.replace('\\n', '<br />
    self.consoleWidget.appendHtml(output)

# Event handle fired when there is new stdout or stderr from SIMF
def console_write(self):
    output = str(self.simfProcess.readAll(), encoding='utf-8')
    self.console_write_line(output)

# Fired by the OK button on the sudo dialog
def start_capture(self):
    # Retrieves the password via a QDialog
    self.passprompt = PasswordWindow()
    self.passprompt.exec_()  # Wait for the password dialog to finish
    if self.passprompt.rejectstat:  # Cancel was pressed
        return
    password = self.passprompt.passLine.text()  # Grab the password

    env = QProcessEnvironment.systemEnvironment()
    self.simfProcess.setProcessEnvironment(env)
    self.simfProcess.setWorkingDirectory(Config.lepton_grabber_working_dir)
    self.simfProcess.setProcessChannelMode(QProcess.MergedChannels)
    # Note this is a kinda hacky way to get the script to execute
    # with sudo permissions, likely a better way to do this at the system
    # level
    self.simfProcess.start(Config.bash_path)
    self.simfProcess.writeData(("printf -v pw "%q\n" "\n"
        + password + "\n\n".encode('utf-8'))
    self.simfProcess.writeData(("echo $pw | " + "" + Config.sudo_path + "\n"
        + Config.python_path + "" + 
        " frame_grabber.py"
        + " --dbg_interval "
        + str(Config.dbg_interval) + 
        (" " --dbg_png" if Config.dbg_png
        else ") + 
        " --dbg_ffc_interval "
        + str(Config.dbg_ffc_interval) + 
        " --dbg_capture_count 
        + str(Config.dbg_capture_count) + 
        " --dbg_serial_csv "
        + str(Config.dbg_serial_csv) + 
        " --dbg_capture_count "
        + str(Config.dbg_serial_csv) + 
        " --dbg_serial_csv "
        + self.current / Config.dbg_capture_count) * 100)
def main():
    config = Config()
    app = QApplication(sys.argv)
    mainwindow = MainWindow()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
if __name__ == '__main__':
    main()

B.1.5 filehandlers.py

import os
from PyQt5.QtCore import QThread, pyqtSignal
from watchdog.events import PatternMatchingEventHandler
from datetime import datetime
from .config import Config
from watchdog.observers import Observer

# Computes where the lepton grabber is currently dumping output
class FileHandlerUtils:
    @staticmethod
    def compute_current_data_dir():
        today = datetime.utcnow()

        return Config.lepton_grabber_working_dir + '/' + today.strftime("%y-%m-%d")

class ImageThread(QThread):
    new_image = pyqtSignal(str)

class ImageHandler(PatternMatchingEventHandler):
    patterns = ["*.png"]

    def __init__(self, event_thread):
        super().__init__()
        self.event_thread = event_thread

    def on_created(self, event):
        self.event_thread.new_image.emit(event.src_path)

    def run(self):
        datadir = FileHandlerUtils.compute_current_data_dir()

        if not os.path.exists(datadir):
            # Wait for lepton-grabber to make the directory rather than failing here or waiting for the data directory to be created just create it ourselves
            os.mkdir(datadir)

        observer = Observer()
class LicorThread(QThread):
    new_licor = pyqtSignal(float)

class LicorHandler(PatternMatchingEventHandler):
    patterns = ["*.txt"]

    def __init__(self, event_thread):
        super().__init__()
        self.event_thread = event_thread

    def on_created(self, event):
        file = open(event.src_path, "r")
        output = float(file.read().strip())
        file.close()
        self.event_thread.new_licor.emit(output)

def run(self):
    datadir = FileHandlerUtils.compute_current_data_dir()

    if not os.path.exists(datadir):
        # Wait for lepton-grabber to make the directory rather than failing
        # here or waiting for the data directory to be created just create
        # it ourselves
        os.mkdir(datadir)

    observer = Observer()
    # FIXME: When this ticks over to the next day it fails to update
    observer.schedule(self.LicorHandler(self), path=datadir)

    observer.start()
    observer.join()
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QDialog, QDialogButtonBox

class Config:
    # TODO: Comment generated config file
    # TODO: Recover gracefully from faulty config
    config_name = str(Path.home()) + '/.simfgui.ini'

    """ READ ME BEFORE EDITING BELOW
    These are the **default** values, if you want to change
    configuration values use the GUI or modify the config file named
    in the variable above and relaunch the program!
    
    If you don't have a config file, just run the program for the first
time to generate the default file.
    ""

    # paths
    lepton_grabber_working_dir = str()
    python_path = str()
    sudo_path = str()
    # Note that bash can be replaced by other compatible shells
    # All I use is echo, quit, printf, and the piping command "|
    bash_path = str()

    # lepton-grabber launch options
    dbg_interval = int()
    dbg_png = bool()
    dbg_fcc_interval = int()
    dbg_capture_count = int()
    dbg_serial_csv = bool()
    dbg_lepton_set = int()
    dbg_testmode1 = bool()
    dbg_print = bool()
    dbg_ser_noavg = bool()
    dbg_no_serial = bool()

    @staticmethod
def defaults():
        Config.lepton_grabber_working_dir = "lepton-grabber"
        Config.python_path = "/usr/bin/python3"
        Config.sudo_path = "sudo"
        Config.bash_path = "bash"

        Config.dbg_interval = 10
Config.dbg_png = True
Config.dbg_ffc_interval = -180
Config.dbg_capture_count = 720
Config.dbg_serial_csv = True
Config.dbg_lepton_set = 7

# Note that this block of config values is named for whether the
# flag they correspond to is on or not, so it logically could seem
# backwards
Config.dbg_testmode1 = False
Config.dbg_print = False
Config.dbg_ser_noavg = False
Config.dbg_no_serial = False

@staticmethod
def read_config(parser):
    parser.read(Config.config_name)

    # Get Paths
    Config.lepton_grabber_working_dir = parser.get('Paths', 'lepton_grabber_working_dir')
    Config.python_path = parser.get('Paths', 'python_path')
    Config.sudo_path = parser.get('Paths', 'sudo_path')
    Config.bash_path = parser.get('Paths', 'bash_path')

    # Get lepton config
    Config.dbg_interval = parser.getint('LeptonGrabberLaunchOptions', 'dbg_interval')
    Config.dbg_png = parser.getboolean('LeptonGrabberLaunchOptions', 'dbg_png')
    Config.dbg_ffc_interval = parser.getint('LeptonGrabberLaunchOptions', 'dbg_ffc_interval')
    Config.dbg_capture_count = parser.getint('LeptonGrabberLaunchOptions', 'dbg_capture_count')
    Config.dbg_serial_csv = parser.getboolean('LeptonGrabberLaunchOptions', 'dbg_serial_csv')
    Config.dbg_lepton_set = parser.getint('LeptonGrabberLaunchOptions', 'dbg_lepton_set')
    Config.dbg_testmode1 = parser.getboolean('LeptonGrabberLaunchOptions', 'dbg_testmode1')
    Config.dbg_print = parser.getboolean('LeptonGrabberLaunchOptions', 'dbg_print')
    Config.dbg_ser_noavg = parser.getboolean('LeptonGrabberLaunchOptions', 'dbg_ser_noavg')
    Config.dbg_no_serial = parser.getboolean('LeptonGrabberLaunchOptions', 'dbg_no_serial')
@staticmethod
def write_config(parser):
    parser.add_section('Paths')
    parser.set('Paths', 'lepton_grabber_working_dir', Config.lepton_grabber_working_dir)
    parser.set('Paths', 'python_path', Config.python_path)
    parser.set('Paths', 'sudo_path', Config.sudo_path)
    parser.set('Paths', 'bash_path', Config.bash_path)

    parser.add_section('LeptonGrabberLaunchOptions')
    parser.set('LeptonGrabberLaunchOptions', 'dbg_interval', str(Config.dbg_interval))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_png', str(Config.dbg_png))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_ffc_interval', str(Config.dbg_ffc_interval))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_capture_count', str(Config.dbg_capture_count))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_serial_csv', str(Config.dbg_serial_csv))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_lepton_set', str(Config.dbg_lepton_set))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_testmode1', str(Config.dbg_testmode1))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_print', str(Config.dbg_print))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_ser_noavg', str(Config.dbg_ser_noavg))
    parser.set('LeptonGrabberLaunchOptions', 'dbg_no_serial', str(Config.dbg_no_serial))

    with open(Config.config_name, 'w') as file:
        parser.write(file)

def __init__(self):
    parser = ConfigParser(allow_no_value=True)
    if os.path.isfile(Config.config_name):
        self.read_config(parser)
    else:
        # Generate new config
        self.defaults()
        self.write_config(parser)
class ConfigEditor(QDialog):
    def __init__(self):
        super().__init__(flags=Qt.WindowStaysOnTopHint)
        ui_file = pkg_resources.resource_filename(__name__, "SettingsDialog.ui")
        uic.loadUi(ui_file, self)
        self.show()
        self.update_configs()

        # Init handlers
        self.buttonBox.accepted.connect(self.apply_settings)
        self.buttonBox.button(QDialogButtonBox.RestoreDefaults).clicked.connect(self.apply_defaults)

    def update_configs(self):
        self.leptonWorkingDir.setText(Config.lepton_grabber_working_dir)
        self.pythonPath.setText(Config.python_path)
        self.sudoPath.setText(Config.sudo_path)
        self.shellPath.setText(Config.bash_path)
        self.capInterval.setValue(Config.dbg_interval)
        self.ffcInterval.setValue(Config.dbg_ffc_interval)
        self.captureCount.setValue(Config.dbg_capture_count)
        self.leptonSet.setValue(Config.dbg_lepton_set)
        self.pngEnable.setChecked(Config.dbg_png)
        self.pngDisable.setChecked(not Config.dbg_png)
        self.csvEnable.setChecked(Config.dbg_serial_csv)
        self.csvDisable.setChecked(not Config.dbg_serial_csv)
        self.testmodeEnable.setChecked(Config.dbg_testmode1)
        self.testmodeDisable.setChecked(not Config.dbg_testmode1)
        self.printEnable.setChecked(Config.dbg_print)
        self.printDisable.setChecked(not Config.dbg_print)
        self.avgEnable.setChecked(Config.dbg_ser_noavg)
        self.avgDisable.setChecked(not Config.dbg_ser_noavg)
        self.serialEnable.setChecked(Config.dbg_no_serial)
        self.serialDisable.setChecked(not Config.dbg_no_serial)

    def apply_settings(self):
        parser = ConfigParser(allow_no_value=True)
        # Set the settings
        Config.lepton_grabber_working_dir = self.leptonWorkingDir.text()
        Config.python_path = self.pythonPath.text()
Config.sudo_path = self.sudoPath.text()
Config.bash_path = self.shellPath.text()

Config.dbg_ffc_interval = self.ffcInterval.value()
Config.dbg_interval = self.capInterval.value()
Config.dbg_capture_count = self.captureCount.value()
Config.dbg_lepton_set = self.leptonSet.value()

Config.dbg_png = self.pngEnable.isChecked()
Config.dbg_serial_csv = self.csvEnable.isChecked()
Config.dbg_testmode = self.testmodeEnable.isChecked()
Config.dbg_print = self.printEnable.isChecked()
Config.dbg_ser_noavg = self.avgEnable.isChecked()
Config.dbg_no_serial = self.serialEnable.isChecked()

Config.write_config(parser)

def apply_defaults(self):
    parser = ConfigParser(allow_no_value=True)

    Config.defaults()
    self.update_configs()
    Config.write_config(parser)

B.1.7 about.py

import pkg_resources
from PyQt5 import uic
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QDialog
import os

class AboutDialog(QDialog):
    def __init__(self):
        super().__init__(flags=Qt.WindowStaysOnTopHint)

        version_file = open(pkg_resources.resource_filename(__name__, "VERSION"))
        version = version_file.read().strip()
uifile = pkg_resources.resource_filename(__name__, "AboutDialog.ui")
uic.loadUi(uifile, self)

        self.show()
        self.versionLabel.setText("Version: "+ version)
B.1.8 sudo.py

# This file handles the sudo password dialog
import pkg_resources
from PyQt5 import uic
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QDialog

class PasswordWindow(QDialog):
    rejectstat = False

    def __init__(self):
        super().__init__(flags=Qt.WindowStaysOnTopHint)
        ui_file = pkg_resources.resource_filename(__name__, "PasswordDialog.ui")
        uic.loadUi(ui_file, self)
        self.show()
        self.rejected.connect(self.pass_reject)

    def pass_reject(self):
        self.rejectstat = True

B.2 Image Processing

B.2.1 SIMF_imstitch_test.m

L0 = L0_0;
L1 = L1_0;
L2 = L2_0;
L3 = L3_0;
L4 = L4_0;
L5 = L5_0;
L6 = L6_0;
L7 = L7_0;

L1_im = mat2gray(L1);
L5_im = mat2gray(L5);
L6_im = mat2gray(L6);
L4_im = mat2gray(L4);
L2_im = mat2gray(L2);
L0_im = mat2gray(L0);
L3_im = mat2gray(L3);
L7_im = mat2gray(L7);

blankimage = zeros(225,400);
for x = 1:60
    for y = 1:80
        blankimage(x+130,y+120) = L5_im(x,y);
    end
end

figure;
imshow(blankimage);

L6_rot = imrotate(L6_im,25);

figure;
subplot(1,2,1);
imshow(L6_rot);
subplot(1,2,2);
imshow(L5_im);

[szx,szy] = size(L6_rot);

for x = 1:szx
    for y = 1:szy
        if L6_rot(x,y) == 0
            blankimage(x+81,y+65) = blankimage(x+81,y+65);
        else
            blankimage(x+81,y+65) = L6_rot(x,y);
        end
    end
end

figure;
imshow(blankimage);

L4_rot = imrotate(L4_im,-8);

figure;
subplot(1,2,1);
imshow(L5_im);
subplot(1,2,2);
imshow(L4_rot);

[szx,szy] = size(L4_rot);

for x = 1:szx
    for y = 1:szy
        if L4_rot(x,y) == 0
blankimage(x+132,y+190) = blankimage(x+132,y+190);
else
  blankimage(x+132,y+190) = L4_rot(x,y);
end
end
end

figure;
imshow(blankimage);

L0_rot = imrotate(L0_im,-4);
figure;
subplot(1,2,1);
imshow(L5_im);
subplot(1,2,2);
imshow(L0_rot);

[szx,szy] = size(L0_rot);

for x = 1:szx
  for y = 1:szy
    if L0_rot(x,y) == 0
      blankimage(x+75,y+160) = blankimage(x+75,y+160);
    else
      blankimage(x+75,y+160) = L0_rot(x,y);
    end
  end
end

figure;
imshow(blankimage);

L3_rot = imrotate(L3_im,-15);
figure;
subplot(1,2,1);
imshow(L5_im);
subplot(1,2,2);
imshow(L3_rot);

[szx,szy] = size(L3_rot);

for x = 1:szx
  for y = 1:szy
    if L3_rot(x,y) == 0

30
blankimage(x+75,y+235) = blankimage(x+75,y+235);
else
    blankimage(x+75,y+235) = L3_rot(x,y);
end
end

figure;
imshow(blankimage);

L2_rot = imrotate(L2_im,37);

figure;
subplot(1,2,1);
imshow(L5_im);
subplot(1,2,2);
imshow(L2_rot);

[szx,szy] = size(L2_rot);
for x = 1:szx
    for y = 1:szy
        if L2_rot(x,y) == 0
            blankimage(x+5,y+247) = blankimage(x+5,y+247);
        else
            blankimage(x+5,y+247) = L2_rot(x,y);
        end
    end
end

figure;
imshow(blankimage);

B.2.2 SIMF_suntracking.m

%% capture 0

%import
% L1_0 = importdata('18-07-20-16-24-11-0-Lepton-5a4a1d.csv');
% L5_0 = importdata('18-07-20-16-24-11-0-Lepton-8b491d.csv');
% L6_0 = importdata('18-07-20-16-24-11-0-Lepton-074c1d.csv');
% L4_0 = importdata('18-07-20-16-24-11-0-Lepton-53491d.csv');
% L2_0 = importdata('18-07-20-16-24-11-0-Lepton-b84b1d.csv');
% L0_0 = importdata('18-07-20-16-24-11-0-Lepton-f44b1d.csv');
% L3_0 = importdata('18-07-20-16-24-11-0-Lepton-f3491d.csv');
% L7_0 = importdata('18-07-20-16-24-11-0-Lepton-fit491d.csv');
L1_0 = importdata('18-07-20-20-25-11-589-Lepton-5a4a1d.csv');
L5_0 = importdata('18-07-20-20-25-11-589-Lepton-8b491d.csv');
L6_0 = importdata('18-07-20-20-25-11-589-Lepton-074c1d.csv');
L4_0 = importdata('18-07-20-20-25-11-589-Lepton-53491d.csv');
L2_0 = importdata('18-07-20-20-25-11-589-Lepton-b84b1d.csv');
L0_0 = importdata('18-07-20-20-25-11-589-Lepton-f44b1d.csv');
L3_0 = importdata('18-07-20-20-25-11-589-Lepton-f3491d.csv');
L7_0 = importdata('18-07-20-20-25-11-589-Lepton-fc491d.csv');

figure;
subplot(2,4,1);
surfc(L1_0);
title('L1');
subplot(2,4,2);
surfc(L5_0);
title('L5');
subplot(2,4,3);
surfc(L6_0);
title('L6');
subplot(2,4,4);
surfc(L4_0);
title('L4');
subplot(2,4,5);
surfc(L2_0);
title('L2');
subplot(2,4,6);
surfc(L0_0);
title('L0');
subplot(2,4,7);
surfc(L3_0);
title('L3');
subplot(2,4,8);
surfc(L7_0);
title('L7');

% image
L1_im0 = mat2gray(L1_0);
L5_im0 = mat2gray(L5_0);
L6_im0 = mat2gray(L6_0);
L4_im0 = mat2gray(L4_0);
L2_im0 = mat2gray(L2_0);
L0_im0 = mat2gray(L0_0);
L3_im0 = mat2gray(L3_0);
L7_im0 = mat2gray(L7_0);
figure;
subplot(2,4,1);
imshow(L1_im0);
title('L1');
subplot(2,4,2);
imshow(L5_im0);
title('L5');
subplot(2,4,3);
imshow(L6_im0);
title('L6');
subplot(2,4,4);
imshow(L4_im0);
title('L4');
subplot(2,4,5);
imshow(L2_im0);
title('L2');
subplot(2,4,6);
imshow(L0_im0);
title('L0');
subplot(2,4,7);
imshow(L3_im0);
title('L3');
subplot(2,4,8);
imshow(L7_im0);
title('L7');

% [centers, radii, metric] = imfindcircles(L7_im0,[1 50],'ObjectPolarity','bright');
% viscircles(centers, radii,'EdgeColor','b');

%% capture 371

L1_371 = importdata('18-07-20-17-26-14-371-Lepton-5a4ald.csv');
L5_371 = importdata('18-07-20-17-26-14-371-Lepton-8b491d.csv');
L6_371 = importdata('18-07-20-17-26-14-371-Lepton-074c1d.csv');
L4_371 = importdata('18-07-20-17-26-14-371-Lepton-53491d.csv');
L2_371 = importdata('18-07-20-17-26-14-371-Lepton-b84b1d.csv');
L0_371 = importdata('18-07-20-17-26-14-371-Lepton-f44b1d.csv');
L3_371 = importdata('18-07-20-17-26-14-371-Lepton-f3491d.csv');
L7_371 = importdata('18-07-20-17-26-14-371-Lepton-fc491d.csv');

figure;
subplot(2,4,1);
surfc(L1_371);
title('L1');
subplot(2,4,2);
surf(L5_371);
title('L5');
subplot(2,4,3);
surf(L6_371);
title('L6');
subplot(2,4,4);
surf(L4_371);
title('L4');
subplot(2,4,5);
surf(L2_371);
title('L2');
subplot(2,4,6);
surf(L0_371);
title('L0');
subplot(2,4,7);
surf(L3_371);
title('L3');
subplot(2,4,8);
surf(L7_371);
title('L7');

% image
L1_im371 = mat2gray(L1_371);
L5_im371 = mat2gray(L5_371);
L6_im371 = mat2gray(L6_371);
L4_im371 = mat2gray(L4_371);
L2_im371 = mat2gray(L2_371);
L0_im371 = mat2gray(L0_371);
L3_im371 = mat2gray(L3_371);
L7_im371 = mat2gray(L7_371);

figure;
subplot(2,4,1);
imshow(L1_im371);
title('L1');
subplot(2,4,2);
imshow(L5_im371);
title('L5');
subplot(2,4,3);
imshow(L6_im371);
title('L6');
subplot(2,4,4);
imshow(L4_im371);
title('L4');
subplot(2,4,5);
imshow(L2_im371);
title('L2');
subplot(2,4,6);
imshow(L0_im371);
title('L0');
subplot(2,4,7);
imshow(L3_im371);
title('L3');
subplot(2,4,8);
imshow(L7_im371);
title('L7');