Fachstudie

Garden Planner for a Greenhouse

Autonomous Garden Assistant for Smart Homes

Verfasser: Michael Hersam, Jannis Westermann, Philipp Wonner

Institut: Institut für Architektur von Anwendungssystemen, Abt. Service Computing

Prüfer: Prof. Dr. M. Aiello

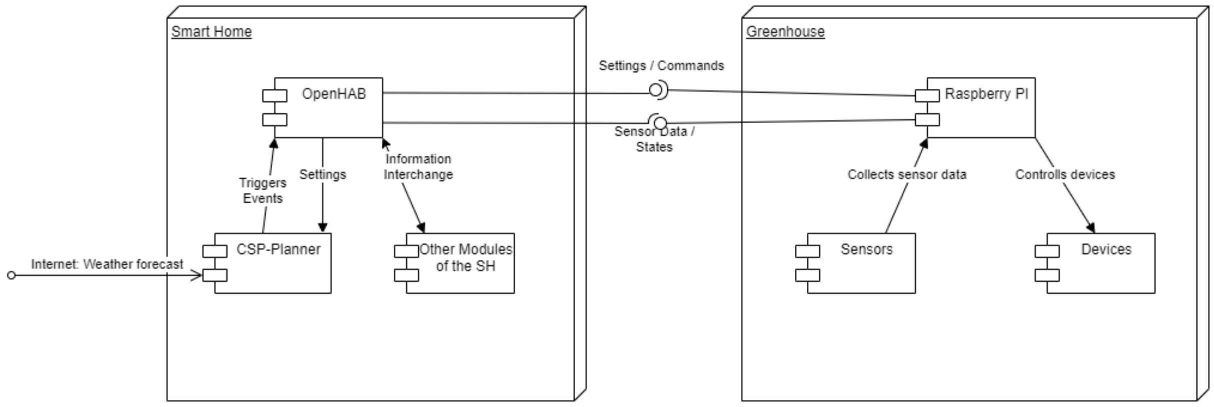
Abgabetermin: 02.12.2020

Table of Contents

Greenhouse

This section will primally describe the function of the greenhouse and its components and how it interacts with the Smart Home System.

Components of the Garden Planner



Smart Home

This is a short summary of the components of the Smart Home that are not part of the greenhouse and how they interact with the greenhouse.

openHAB:

Provides the user interface of the Smart Home System and acts as a connector and interface between the different components of the Smart Home. In this way it provides the UI for settings and displays data and states of the whole system. It is also controlled/trigger by the CSP-Planner that uses openHAB to delegate the tasks to the individual components.

openHAB integration helper:

By default, openHab provides multiple ways of interacting with its components. These include bindings, which are custom modules loaded into openHAB, automation engine modules, which are triggers and conditions for executing automation rules and scripts and a REST interface, among others.

Since planning solutions using CSP-planners are external programs written to solve a specific problem, we decided to write a lightweight java library wrapping the REST interface. Since the interaction is mostly limited to getting and setting variables in openHAB (so-called “items”) we can compromise on other functionality one may get by writing a native openHAB-binding. This way also makes it easily adaptable to changes on both the openHAB as well as the CSP side.

The integration helper provides three main functionalities:
1. Initialization: On initialization the helper reads a config file like this:

```
{
	"address"			: "http://localhost:8080/rest/",
	"items"				: [
		{
			"openHABName" : "gp_soil_hum",
			"alias"		: "soil_humidity",
			"type"		: "Number",
		},
		{
			"openHABName" : "gp_par_water_ml",
			"alias"		: "water_amount",
			"type"		: "Number",
		},
	]
}
```

This config file shall include all openHAB items which are read and/or set by the planner.
The helper makes sure that a connection to the server can be established and all items needed are available. Optionally a name alias can be specified with which the item can be accessed, potentially helping with readability and disparities in naming conventions. The type parameter helps with the correct conversion between the string representations of the REST API and the Java types.

2. Getting and setting items: accessing the values of items is done with the methods:

```
	public double getNumber(String name);
	public String getString(String name);
	public boolean getSwitch(String name);
	etc.
```

Setting the values is done with:

```
	public void setItem(String name, Object value);
```

which also does type checking.

3. Updating schedule: For our CSP planner which needs to recalculate in regular intervals the function

```
	public void startScheduleUpdates(String[] items,
							IfunctionPointer scheduler,
							int interval, int steps);
```

is provided. It takes the names of the items to be updated, the update function, the time interval and the number of steps calculated ahead as arguments. In case one update fails, the previous schedule is used until it gets updated again.

CSP-Planner:

It is the heart of the Garden Planner System. With the weather forecast and data of the Greenhouse it, creates a plan when and how to do the single tasks and jobs to provide the best care for the plants. The CSP-Planner can also be used to solve problems for other components if they are integrated later in the right way. The CSP-Planner uses some settings for goals and constraints from openHAB. In this way openHAB provides the UI of the CSP-Planner. This can be for example: Min- and max light hours for the plants, crop coefficient, min soil moisture, etc.

The planner is written in Java and uses OptaPlanner [1], an AI constraint solver that solves optimization problems efficiently. The planner runs periodically, the time between each run is variable and can be defined by the user, short intervals are recommended to achieve the best results since real world conditions and predicted conditions may diverge. The planner calculates an optimal plan for an up to 96-hour period, and schedules required irrigation, artificial lighting, heating and shutter states. For that, live sensor data and an hourly weather forecasts are used to create plans. Climacell's API [2] is used to get hourly weather forecasts including air temperature, precipitation probability and amounts, wind speed, short wave radiation and dew point values to predict the evapotranspiration (ET_0). Combined with a crop coefficient k_c that depends on the plant and its development stage, an even more accurate approximation can be achieved. For calculating the evapotranspiration, an established agricultural approach is used, which is the FAO Penman-Monteith equation [3]. The original equation does not allow for hourly predictions, so we used a modified version [4, 5]. The planner needs to evaluate potential solutions to find the best one, therefore a list of constraints is defined, that penalize undesired conditions. E.g. the soil moisture leaves a specific range, or the shutters are opened when strong wind gusts occur. The planner is built to be robust to sensor or weather forecast deviations or complete failure. For example, when the system is unable to retrieve the weather forecast, there might be a forecast still saved from earlier runs that covers the upcoming hours that can be used.

[1] https://www.optaplanner.org/

[2] https://developer.climacell.co/v3/reference

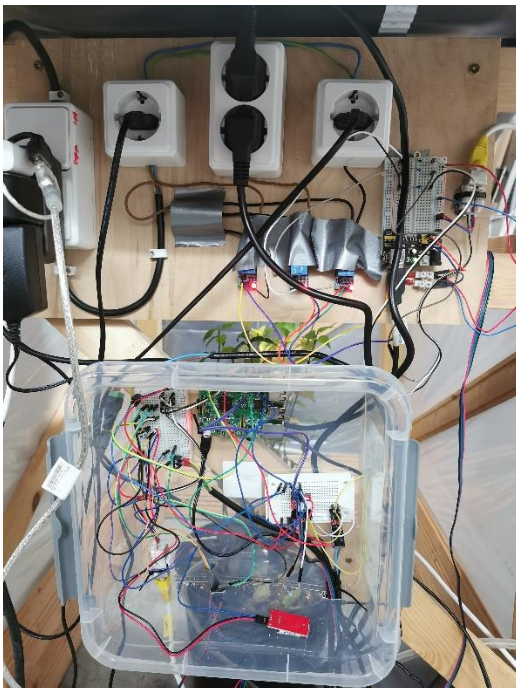
[3] http://www.fao.org/3/x0490e/x0490e01.htm#preface

[4] https://doi.org/10.1016/j.biosystemseng.2014.10.011

[5] https://www.researchgate.net/publication/237412886_Penman-Monteith_hourly_Reference_Evapotranspiration_Equations_for_Estimating_ETos_and_ETrs_with_Hourly_Weather_Data

Greenhouse

Raspberry PI:



The Raspberry Pi is the main element of the greenhouse. It collects the data from the sensors and controls the connected devices. It is connected to openHAB by a TCP-Connection. openHAB can send commands to the Raspberry PI, on the other hand the RaspPI provides the collected sensor data and inner states of the greenhouse system to openHAB. So, in fact the command sequence and sensor data flow look like this:

Command sequence:
CSP-Planner/Human (API/UI) -> openHAB (TCP)-> Raspberry PI (GPIO) -> devices

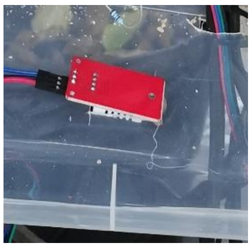
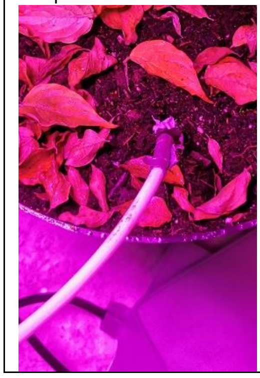
The sensor data flow is the other way round.

The control program is written in python because it offers a good way to program the GPIOs (General Purpose Input/Output) and many libraries for devices and sensors are available in python for the Raspberry OS.

In fact, the Raspberry PI has enough power that openHAB and the CSP-Planner can be executed on it, so for small Smart Home solutions or a stand-alone greenhouse, no Distributed System is needed.

Sensors:

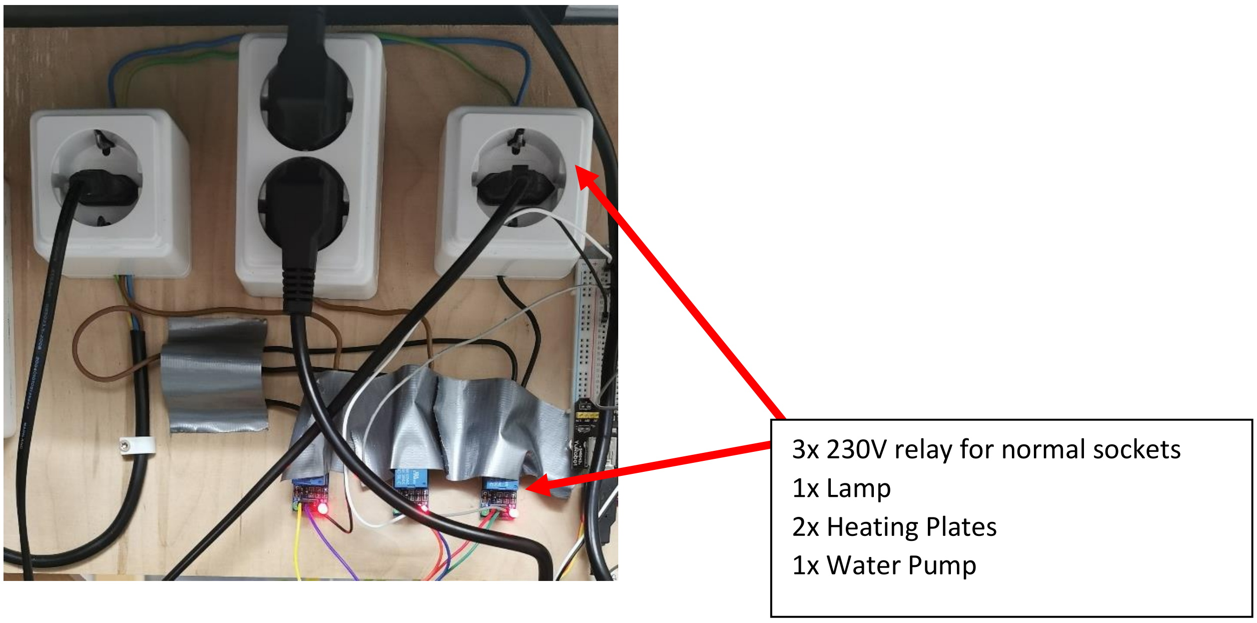
The connected sensors are a Lux-Meter (light sensor), Temperature/Humidity Sensor and a Soil Moisture Sensor. All sensors data are exponential smoothed, mainly because the Temperature/Humidity Sensor has some outliers. As self-check, if a sensor is not reachable form the system three times in a row, a warning is sent to the openHAB system and is marked as unviable there. This is important for the CSP-Planner and for the user.

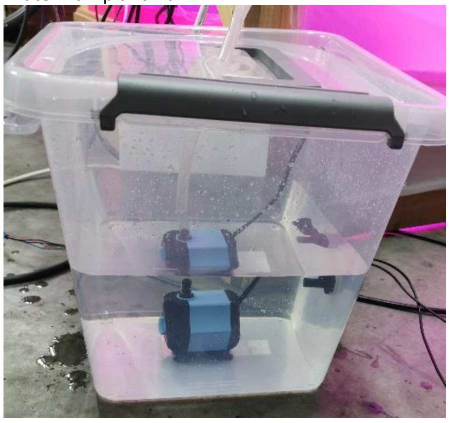
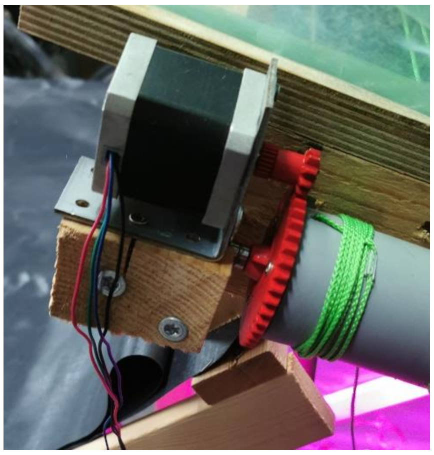
Name/Photo	Protocol	Description
Light Sensor: BH1750 	I2C, using a lib of oskar456 (GitHub) https://gist.github.com/oskar456/95c66d564c58361ecf9f	Gets the Lux of the sunlight. We are using this formular to convert them into w/m^2: 1000 lx = 8,0 W/m^2 In addition, we count the "Sun Hours" to check if the Plant get enough or too much sun. A Sun Hour is count if the light is above 120 w/m^2, that is about 15000 lux It is covered with a semitransparent plastic (filter factor ~2.75) and silicone to protect the sensor from the weather conditions It provides: Light in Lux, Light in w/m^2 Sun Hours
Temperature/Humidity Sensor: DHT22 	Some 1-Wire protocol. Using Adafruit_DHT lib. https://learn.adafruit.com/dht	This sensor gives the temperature and the air humidity. It is placed in the case of the raspberry on the back of the greenhouse to protect the sensor from direct sunlight and rain that will cause wrong temperature and humidity data. It provides: Temperature in °C Air Humidity in %
Soil Moisture Sensor: Chirp! I2c Soil Moisture Sensor 	I2C, using lib from Miceuz from GitHub https://github.com/Miceuz/PlantWateringAlarm	This sensor is placed in the soil and should be calibrated if the potting soil is changed, because each soil has other saturation properties. In this case a "Switch" in openHAB can be used after a full watering to set that saturation point as 100% It provides: Soil moisture in %

(https://www.adafruit.com/product/1965)		
Level Indicator	Simple passthrough switch	With the level indicator openHAB is informed if the tank is empty. If a watering command is received, it will only be executed if the tank is not empty to protect the pump from damage.

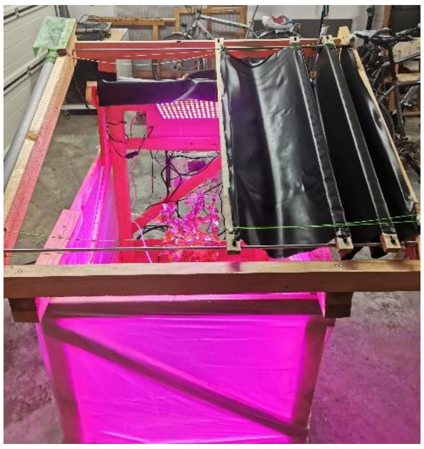
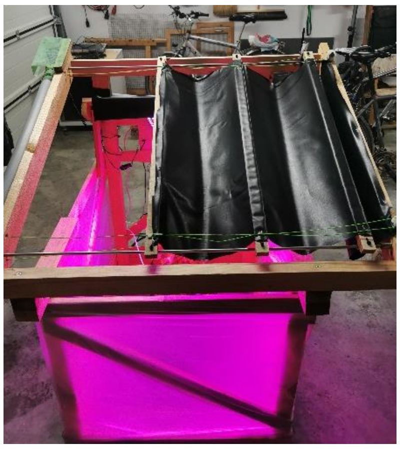
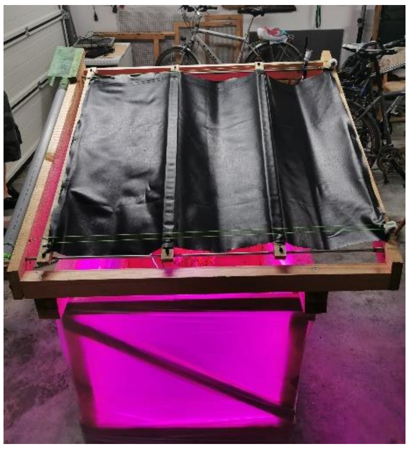
Devices:

The devices used in the greenhouse are a shutter, heating plates, a lamp and a water pump. Different to the more complex control of the shutter, the heating plates and the lamp are just activated through a 230V relay.



Name/Photo	Protocol/control	Description
Heating Plates	Controlled by relay	There are two heating plates with 15W each to protect the plant from freezing
Lamp	Controlled by relay	A LED grow lamp with blue and red LEDs, that matches the needed light spectrum plants need to grow. The lamp has about 35 Watt. It does not provide the power of a clear sky sun but gives good support on dark cloudy days.
Water Pump and Tank 	Controlled by relay	The water flow must be calibrated once. In openHAB as UI you can set the flow in ml/min. After the calibration, the CSP-Planner can set the needed amount of water in ml and start watering.
Shutter 	Stepper Motor Driver, similar to the A4988.	The stepper motor is driven by a A4988 clone and 12V. We use a direct control through the GPIOs. (self-written library) Through a rope system and a linear glide bearing, the segment shutter is opened and closed.

Shutter in action:

Problem Analysis and Discussion

Multithreading

The control program for the sensors and devices on the Raspberry Pi has to manage different task parallel. For example, the TCP-Socket has to wait for receiving messages and commands from openHAB, while devices (such as the shutters) have to be controlled and sensor data is collected. First, we had the Problem, that waiting for the next TCP-Message, blocked the whole System, while longer Tasks (watering or shutter control) blocked the TCP-Socket. To solve this Problem, we used different threads for these tasks.

TCP-Connection between the Components

As we wanted to create a system, that will work also as a Distributed Smart Home System, we decided to use a TCP-Connection between openHAB and the control program. As described in chapter “Multithreading” we had to solve the problem of blocking the other parts of the control program by the TCP-Socked. For this we used the open source “TCPCom library” which brought the benefit of an observer pattern for received messages. To distinguish messages that are bigger as a TCP-Package, the Message is expected to be null-terminated as strings in “c” from the TCPCom module. But openHAB isn’t allowing to set a Zero-Character. Not knowing about that different patterns for the TCP-Connection in the different modules, we first had some unpredictable behavior. So we had to modify that in the library and exchanged the termination symbol with a “\n”-character. Further we used JSON to pack our data.

Sensors and Devices

As we didn’t have much contact to electrical engineering and microcontrollers, we had to figure how to address the sensors and devices. So, we get into the I²C protocol and the other, partially not official and poorly documented, protocols.

And we had to deal with some foibles of some of the devices:

Relays: They have a big power consume of about 70-200 mA, that the Raspberry could not handle, so we had to give them an additional power supply.

Moisture sensor: The address of the I²C-Connection is not sufficient protected. So sometimes it will change its address and isn’t reachable anymore. The newer series of the sensor has a better protection, so for further projects we recommend to use Sensors with the firmware 2.6 (instead 2.4)

Light sensor: The light sensor is calibrated but to get it waterproof and give it diffuse light, we used a white semi-transparent plastic foil. So we had to measure the filter factor of the foil. In addition, we had to get a formular to calculate w/m^2 from lux for the formular for the evapotranspiration. Because of the different wavelengths of the light a pyranometer is used to measure w/m^2. But if only one light source is measured, it is possible to calculate the right factor from a luxmeter. So we measured about 120-130 klx and used the information, that clear sky sunlight in summer is about 1000 w/m². That fitted verry well the formula of 1000 lx = 8 w/m², we found in an internet forum (https://www.mikrocontroller.net/topic/454389).

But all in all, in a long-term test of five days, the Greenhouse System worked verry well. Once the temperature sensor and the soil moisture sensor failed. And one relay didn’t switch after a while.

Construction of the Shutters: One big issue outside of the domain of computer science was the construction of the shutters. Ready-to-buy solutions were way too expansive, so we had to design our own concept. So, we decided to use a stepper motor and use linear glide bearings. With rows the shutter is pulled in both directions. Unfortunately, the motor was not strong enough to pull the whole construction directly, so we had to build a gear from “Fischertechnik” parts. One issue, that occurred and may get a real problem, is that the shutter construction gets too stiff from rust and soiling, despite that we used stainless steel. Because of this we had to adjust the range of movement of the sliders and the setting of the stepper motor. Adding some kind of covers at the side of the bearings should mitigate this and periodically oiling the rods and bearings should also help.

Cold Weather: Because of its open construction the water container and irrigation tube are not protected against the cold. While the water tank with the pump inside showed only surface level ice at around -5 degrees celcius, the tube froze completely shut. To prevent the freezing of the water tank, a heating pad can be placed underneath it. To prevent the tube from freezing it would need to be mounted in a way which automatically drains any excess water that does not reach the plants. On the other hand, a greenhouse like our model greenhouse isn’t used to grow plants during the winter, so it is recommended to get the greenhouse winterproof. For this all water tubes (even for e.g. Sprinklers, etc.) should be made empty (as every garden has to be prepared for winter) and bearings should be oiled.

Crashes: Sometimes, the Raspberry Pi crashed. This happened at least if openHAB, the control program, the browser (for example for checking data in openHAB) and the development environment ran at the same time. We think it is a leak of memory, our Raspberry Pi 3+ has only 1 GB of ram. So, it would be recommended to use a Raspberry Pi 4 instead. But this happened just a few times.

Evapotranspiration Formula

For now, we use an established formula, the FAO Penman-Monteith equation. This formula is used in big agriculture Farms. It remains to be seen if this fits well to our model greenhouse, especially with potted plants. Here it could be a good approach to create a self-learning algorithm that automatically adopts to the specific soil and plants. This can be compared in follow up project as a bachelor thesis.

Costs and Effort Calculation

Overall, we spend about 300€ and nearly a full month of work, each, into the project.

Listing of the costs:

Raspberry Pi	40€
Sensors and devices	120€
Parts for the shutter	50€
Parts for the greenhouse (screws, wood, sockets, foil)	60€
Miscellaneous Parts (Wire etc.)	40€
	310€

If we calculate the costs to extend an existing greenhouse with our system, it can be much cheaper. But this depends on the existing devices. If a watering system (as sprinklers), an electric shutter and Lamps exist, that just have to be integrated and controlled by the system, we estimate that the costs can be reduced to about 200€.

In a long-term test, different treatments (no shutter, no artificial lights and/or heating) can reveal what types of devices and automated treatments have the highest gain and can show if there are parts in our system that will have a minor effect on the plants. So, in case there are non-rentable devices/sub-systems, they can be omitted to reduce the costs.

Integration of an existing Garden or Greenhouse

As estimated in the chapter “Costs and Effort Calculation”, the costs can be significant less in an existing garden or greenhouse, depending what devices exist. But the costs can also grow and may won’t scale linearly if big shutters or heating systems have to be integrated.

Sprinklers can easily be integrated with magnetic valves and electric shutters can be controlled by relay. If it is a closed greenhouse, the system can work without the shutters, but in this case the maximum light can’t be controlled. For the most plants, this wouldn’t be a big problem, as they also grow under free sky.

Gain and Benefits

The gain and benefits are not easy to predict. In small size, for private use, we expect to save about half an hour of work, each day. But the greater advantage will be the freedom to leave the garden unattended for a few days, like on a holyday trip, while the system takes care and the garden can be checked online with openHAB. With the heating plates, plants will be protected from frost during spring and the early summer months.

In addition, the water costs can be reduced. The system will check, if it will rain, before the moisture level will sink under a critical level. This could be a big benefit if the weather is verry changeable, especially for farmers. This can be checked in a long-term test, here it should be possible to track how often our system decides to wait for rain, instead watering the plants “stupidly” on a chosen and hard-set point of time or moisture level.

In addition, we expect a better growth rate and healthier plants, because our system can react faster to changing weather conditions or unhealthy moisture levels as a “manual” care of the garden. This also can be proofed in a long-term test, as the resulting plants are compared to manual treated plants.

Attachment

In our Dropbox, we stored some videos, that show our greenhouse system in action. We also stored some data, we collected. Unfortunately, we ran out of time, so we couldn’t do an evaluation yet, nevertheless we want to attach the data because its interesting what the system had planed and how the system reacts to changing conditions.
(Link: https://www.dropbox.com/sh/6kb4ngdebf5j2dt/AABGX6EIw6ap2fKY3ucUzbK0a?dl=0)