

An Automated Platform For High Throughput Fluorescence Microscopy

Thomas Adams^{1,2}, Curtis Hauck¹, Cody Gregg¹, Trent Ruiz¹, Andrew Lockwood³, Maximilian Seligman^{1,4}, Brian Rasnow^{1,2}
¹Applied Physics, ²Mechatronics Engineering, ³Computer Science, ⁴Biotechnology/Business Administration

Scan QR code to access our site



Introduction

This project demonstrates automated [high-throughput microscopy](#) using [MATLAB](#) as the control software. MATLAB synchronizes a [Lumascop 500](#) inverted fluorescent microscope, a stepper motor attached to its focus knob that automates focus through an [Arduino](#) microcontroller, and a KiNEDx KX-300 laboratory robot. The objective is to record a stack of images in different focal planes ("z-stack") from every well in a stack of microtiter plates.

The two main challenges for lab automation are to find/make devices that operate reliably without any human intervention, and to integrate these devices so they synergistically execute the experiment. The closest commercial solutions we know of are a [Robotic well-plate loading system](#) sold by Prior Scientific and an [ImageXpress](#) and robot sold by Molecular Devices (see References). Each of these systems cost >\$100,000. Our solution is unique because we modified a manual-focus Lumascop 500 for autofocus, and we used one robot for all plate movements, both between stacks and traversing wells. Eliminating the redundancy of an automated stage and automated microscope makes our system simpler, more compact, and less expensive.

This poster describes the design, construction, and validation of the automated microscopy platform.

Methods

Software Architecture

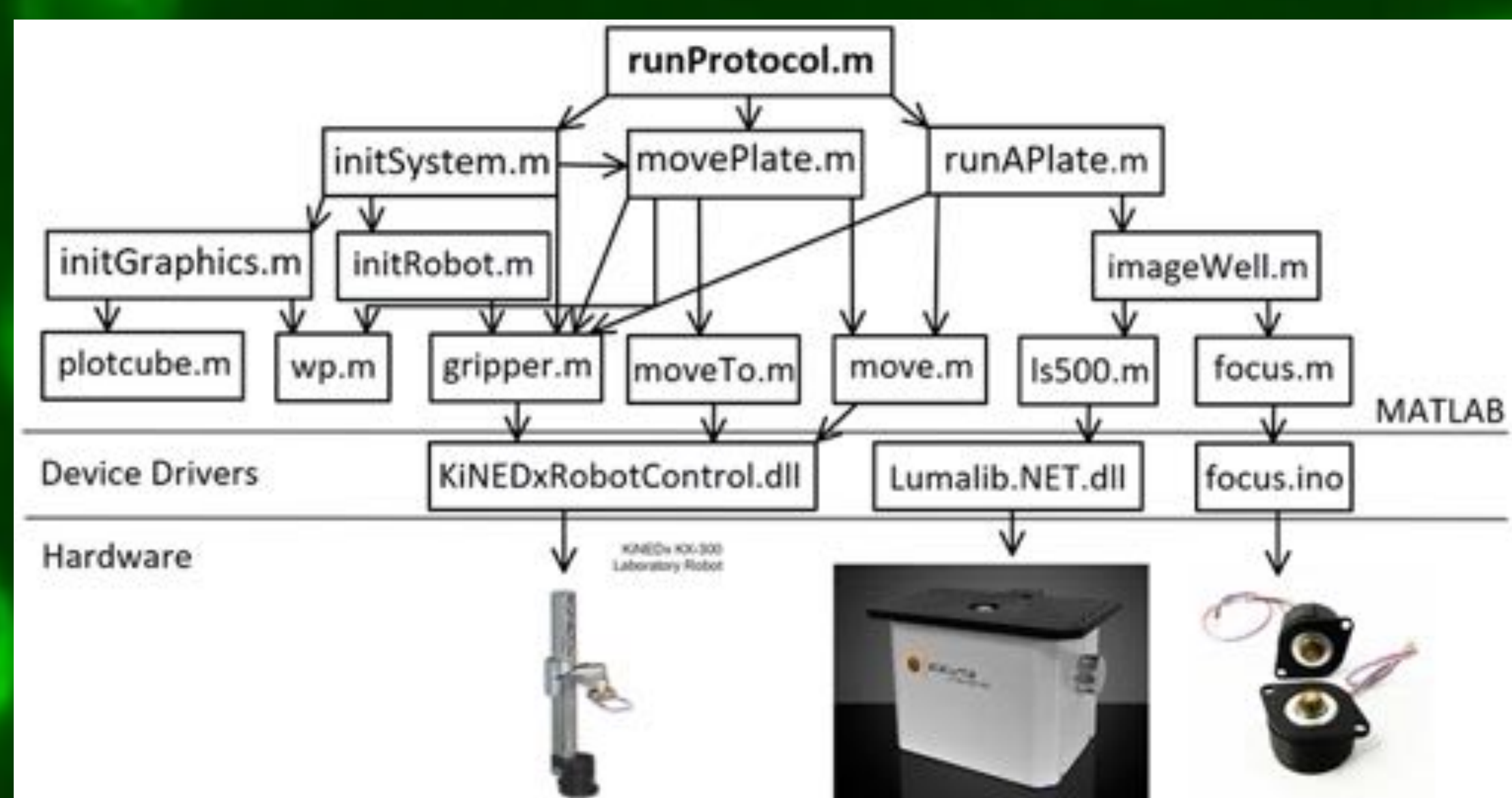
We implemented the control software in ~500 lines of MATLAB spread across 14 files, and ~80 lines of Arduino C controlling the focus motor. Most of these files are functions, with private workspaces, some utilize persistent variables storing serial ports, waypoint coordinates, etc. A few are scripts that share variables in a "public" workspace. Global variable, `r2` is a structure that contains information about robot location, plate number, simulation mode, number of focus planes, etc. A collection of waypoint coordinates and names was implemented in `wp.m`. MATLAB communicates with the robot and Lumascop through dynamically linked libraries (DLL's) provided by their respective vendors. DLLs provide a generic interprocess communication interface for Windows operating systems. The focus motor interface was through an Arduino Uno's USB port. The software architecture was hierarchical, with the highest-level script called `runProtocol.m`, which consists *entirely* of the following code:

```
1 global r2
2 initSystem();
3 for i=1:r2.plate.numplates
4     movePlate('instack');
5     movePlate('LS');
6     runAPlate(num2str(i));
7     movePlate('outstack');
8 end
9 fclose(r2.fp); %clean up
```

This script mixes built-in MATLAB commands with commands we wrote to implement other parts of the protocol, as listed below:

- `initSystem.m` initializes fields of the global `r2` variable, queries the user for the desired simulation mode, output directory, and then initializes the graphics (calling `initGraphics.m`), and connects MATLAB to the robot's DLL.
- `runAPlate.m` is a function that moves each well of the plate sequentially over the Lumascop and calls `imageWell.m`, which executes focus (`focus.m`), imaging (`ls500.m`), and saves data.
- `movePlate.m` executes sequences of moves, gripper actions, log file updates, and housekeeping, e.g., as the `activePlate` is put into `outstack` and a new `activePlate` is retrieved from `instack`.

The collection of 14 functions and scripts we wrote, and how they intercommunicate with each other, with the DLL's and Arduino, and in turn with the hardware, are shown below using an Architecture Interconnect Diagram (AID).



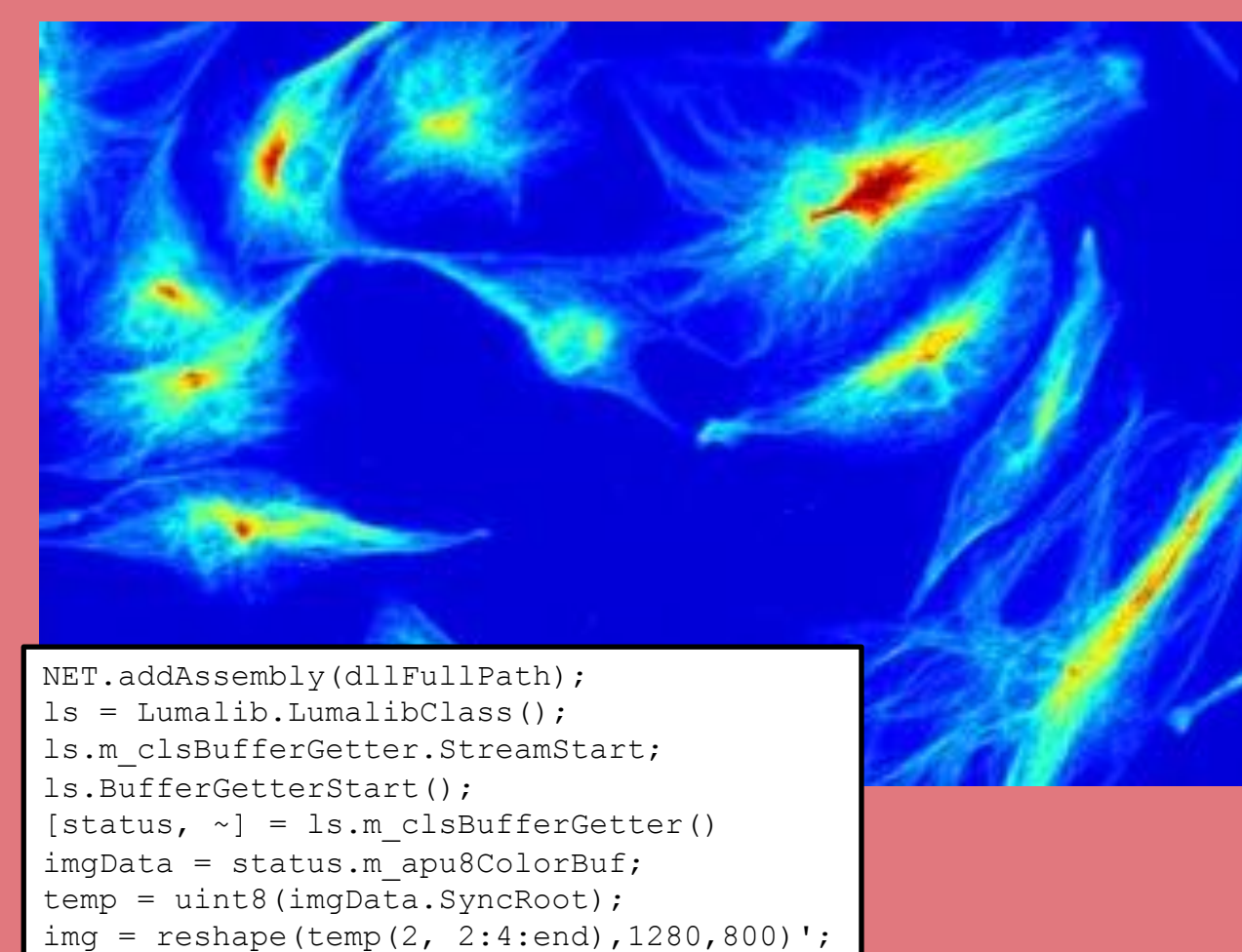
KiNEDx Laboratory Robot

We used a KiNEDx KX-300 robot, a legacy version of the [Peak Robotics KX2-500](#) designed for laboratory automation. This robot has four motorized axes and a pneumatic gripper for [microtiter plates](#). The robot is mounted on a tilt kill switch to prevent damage should a major collision occur. A large emergency stop button was wired in series with robot power for additional safety. The robot software supports a graphical user interface "Teach Pendant" that we used to determine waypoints in robot-centric coordinates and saved them in the function `wp.m`. To run our protocol, the robot was initialized from MATLAB via `net.AddAssembly` (`KiNEDxRobotControl.dll`). The resulting handle to the DLL was saved in a global variable called `r2.dll`. MATLAB communicates with the robot in the following functions: `initRobot.m`, `move.m`, `moveTo.m`, and `gripper.m`. For example, `move.m` in the following lines translating MATLAB variables into the format expected by the DLL:

```
27 sMoveDist = sprintf('%g %g %g %g %g %g %g %g', deltaPosition);
28 result = r2.dll.MoveRelativeCartesian(sMoveDist, vel, 50, waitForDone);
where deltaPosition is a 4-component vector in robot coordinates that specifies a relative move.
```

Lumascop Digital Microscope

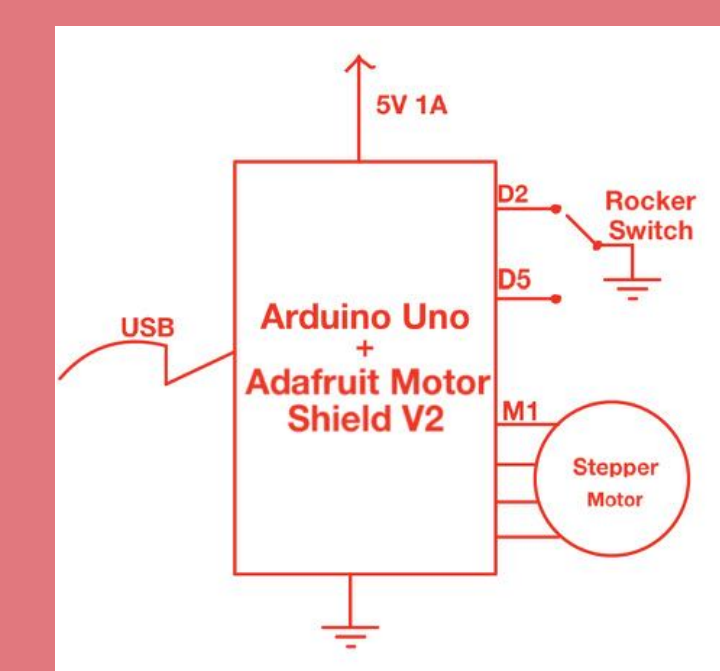
The automated system was prototyped with an Etaluma [Lumascop 500](#) (LS500) inverted fluorescence microscope, while biology and assay development are underway in parallel using our [Lumascop 620](#) (LS620). The latter offers 3 color epifluorescence, improved optics, and a DLL interface that has been demonstrated to work with MATLAB ([Seligman M & Rasnow B, 2020](#)). The LS500 was initialized by launching [Lumaview](#), and subsequently controlled through a legacy Lumascop 500 SDK file, `Lumalib.NET.dll`. Initialization and manipulation of the raw data stream is not straight forward (see code at right). While we were able to return several images, the software tended to crash. To support a simulation mode, images were returned from a previously saved z-stack taken with an LS620. The LS620 interface also enables MATLAB to cycle on and off each color LED, thereby reducing fluorescence bleaching, as well as permitting up to 3 color fluorescence image acquisition.



```
NET.addAssembly(dllFullPath);
ls = lumalib.LumalibClass();
ls.m_clrBufferGetter.StreamStart();
ls.BufferGetterStart();
(status, ~) = ls.m_clrBufferGetter();
imgData = status.m_apo8ColorBuf;
temp = uint8(imgData.SyncRoot);
img = reshape(temp(2, 2:4:end), 1280, 800);
```

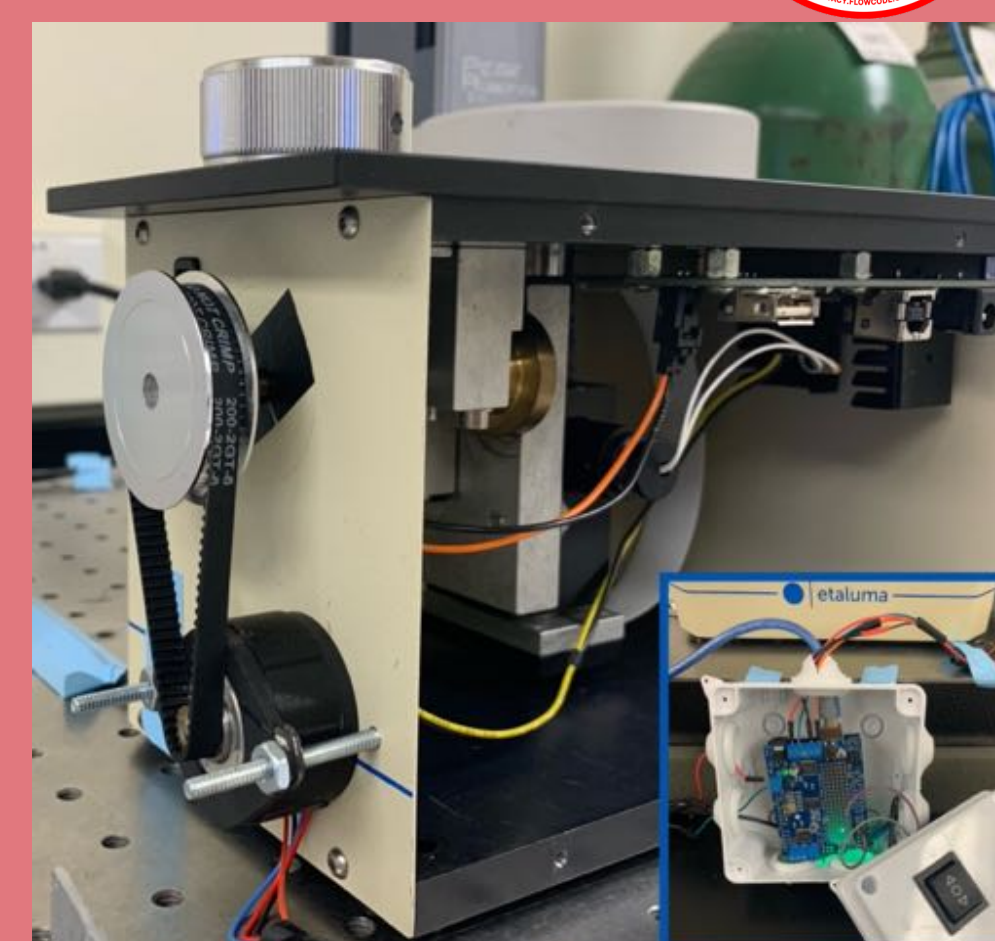
Focus Motor and Arduino Controller

Microscopes have narrow depth of field and microtiter plates are often not flat, therefore fine focusing on each well is a key requirement. We modified a manual-focus Lumascop by bolting a small stepper motor to its case and replaced the focus knob with a 60-tooth gear and belt. The stepper motor was controlled by an [Arduino Uno](#) and an [Adafruit Motor Controller V2](#) shield ([focus.ino](#)). A [rocker switch](#) attached to the Arduino providing natural manual bidirectional control, and [focus.m](#) provided flexible software control of the motor from MATLAB.



The 0.9°/step motor has a 15-tooth gear, and the focus knob has 60 teeth, so it takes 1600 steps per rotation of the focus knob. This is more precise than a typical user can achieve by hand. It also enabled us to use integral steps without necessitating holding current. We release the motor to eliminate vibrations, which might blur images. The gripper is also opened while imaging to eliminate robot vibrations.

The rocker switch executes 10 steps per click and supports continuous movement. The above QR code points to a video demonstrating our automated focus.



MATLAB Simulator

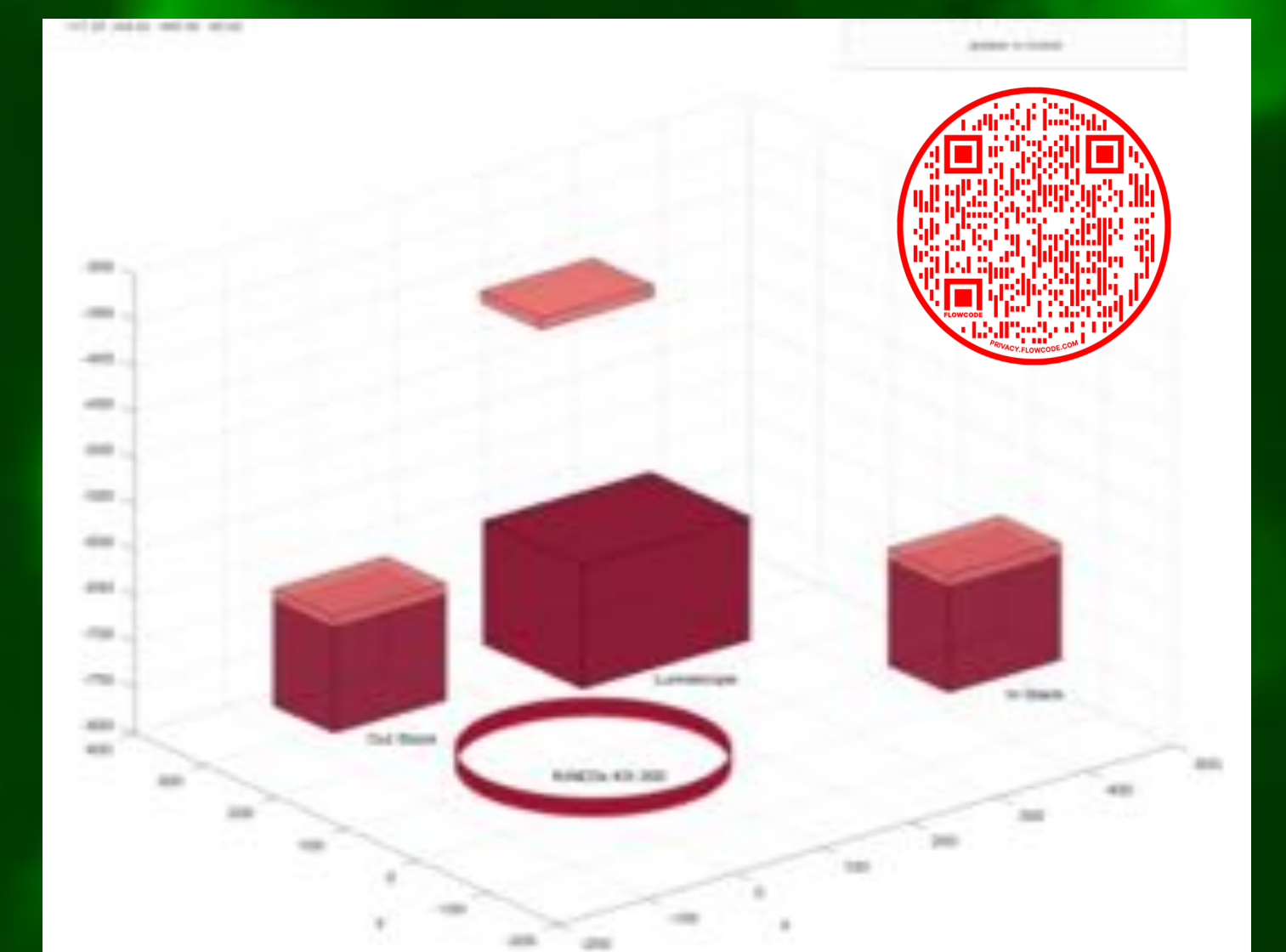
The simulation code begins in `initGraphics.m` which executes at the beginning of the protocol. It calls `plotCube.m`, a function from the public domain we modified to return handles to the graphic object surfaces. The robot-affecting functions `move.m`, and `moveTo.m` simply update the relevant fields of these graphic objects to keep their positions in the simulation window synchronized with the movements of the robot. For example, `moveTo.m` changes all the vertices of all the patches that render the `activePlate` by executing the following:

```
patches = r2.plates(r2.activePlate);
for i = 1:length(patches)
    patches(i).Vertices = patches(i).Vertices + delta;
end
```

Graphics updates, that occur automatically or forced with `drawnow`, move the plate object on the screen. A versatile feature of the simulation is a global Boolean parameter that each hardware driver checks. If `sim` is true, then hardware communication isn't attempted and instead that communication step is simulated -- e.g., by printing a descriptive line to a log file, pausing for the typical duration of the command, etc. Thus, by changing a single Boolean flag, the protocol either runs on the hardware at hardware speed or runs through the identical code without interacting with COM ports, device drivers, etc. that are particular to the laboratory system. This dual mode architecture allows remote work on any computer from anywhere. Any new protocol can be safely run at home without the risk of breaking anything, but then can be immediately implemented on the robot with the flip of a Boolean software switch. The efficiency of the simulator made it a key element in the development of this project.

Results

The figure below shows a photo of the automated system and a snapshot of the simulation, at the same moment when a plate is being transported above the Lumascop by the robot. The QR code points to a video showing the simulator dynamics.



Profile Summary
 Generated: 30-Apr-2022 14:11:30 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
runProtocol	1	3012.166 s	0.010 s	
runAPlate	3	2938.134 s	0.065 s	
imageWell	288	2736.952 s	512.052 s	
focus	2016	1967.780 s	29.069 s	
ls500	1728	453.039 s	453.961 s	
gripper	586	129.110 s	128.955 s	

How to Run a High Throughput Screen

- Place the microtiter plates to image in the input stack and press them flush against the guide bar.
- Open the pneumatic pressure source and verify ~85psi regulated pressure to actuate the gripper.
- Launch Lumaview and set the LED brightness, exposure, and gain appropriate for your sample. Set focus to the bottom of the z-stack range.
- Launch MATLAB. Edit `initSystem.m` to configure the number of plates, focal planes, and focus step.
- Visually verify the deck is clear and orderly before starting.
- Execute `runProtocol`, set simulation mode to false, and name the log file.

Conclusions and Summary

We have achieved the following:

- We modified a manual-focus Lumascop to motorized focus.
- Used one robot for all plate movements, both between stacks and traversing wells.
- Created a flexible and powerful lab automation control architecture in MATLAB that:
 - Controls diverse instruments, including laboratory robots, microscopes, and Arduinos.
 - Concurrently displays a graphical representation of the apparatus.
 - Does this with ~500 lines of MATLAB code containing ~250 function calls!
- Constructed an automated high throughput microscopy system from ~\$50K of equipment that outperforms commercial systems costing more than \$100K.

Next steps are:

- Execute a high throughput phenotypic screen of yeast clones (using an LS620).
- Make user interface improvements targeting ease of use by non-programmers.
- Implement intelligent image processing algorithms to the data, including machine learning.

References and Notes

- See the [Wiki version of this poster](#) for more cost data, source code, and additional information. (adjacent QR code)
- [Lumascop 500](#) and [Lumascop 620](#) are products of [Etaluma, Inc.](#) (which Rasnow is a co-inventor and co-founder)
- KiNEDx robot is a product of [Peak Robotics](#)
- MATLAB is from [www.mathworks.com](#)
- [Arduino Uno](#) and [Adafruit Motor Shield V2](#) were used
- Seligman M, Rasnow B (2020) [Creating a flexible user interface in MATLAB for a Lumascop\(TM\) microscope](#). 12th Annual SAGE Student Research Conference, CSUCI, describes a robust MATLAB interface to a LS620.
- The Lumascop image is of [FluoCells™ Prepared Slide #2 \(BPAE cells with Mouse Anti-α-tubulin, BODIPY™ FL\)](#).
- The background image is yeast expressing green fluorescent protein (GFP) taken with a LS620 by Prof. H. Tapia.

