Analysis of Student-Problems While Working with Physical Computing Devices

Eric Schätz[a], Lutz Hellmig[b] and Alke Martens[c]
Institute of Computer Science, University of Rostock, Albert-Einstein-Straße 22, 18059 Rostock, Germany

Keywords: Physical Computing, Student Problems, Student Assumptions, STEM.

Abstract: During physical computing events, we have offered to school-students, we have observed different problems students struggle with while working with sensors and actuators of physical computing devices. Whereas some problems have been caused by simple programming errors, some of them were not explainable just by the code. In those cases, programs worked correctly in simulations but not on the real device. In this paper looked at different error origins in context of working with physical computing devices. For further analysis, proposed a model of different stadiums, data passes while a task or problem is solved with a physical computing application. Using this model, located observed issues in order to identify wrong assumptions, made by students.

1 INTRODUCTION

From a number of teachers, we have worked with during physical computing workshops, we got the feedback that they keep running into several problems using physical computing devices in class. Even though they really enjoyed working on that topic during our workshops, they speak of many pitfalls which detain them of integrating physical computing in several situations in their lecture. Some of the stated doubts have also been observed by Schmalfeldt and Przybylla in (Schmalfeldt and Przybylla, 2021). Many teachers also claimed *the usage of physical computing in class needs way more preparation time as a class without physical computing*. Another big concern was *physical computing devices may lead to too many and diverse problems which makes it hard for a teacher to provide help for students and makes class too discerning for their students*. Teachers told us, they use physical computing only where it is explicitly mentioned in the curriculum and if though, only in a very basic way: For example they avoided using the calliope(Calliope, 2022) with extentiones such as the Callibot (Scheglmann, 2023). As we noticed that those concerns where typically expressed by teachers which did not study computer science and were struggling with some of the given tasks themselves, this feedback led us to the question how upcoming student issues can be systematized and on which false assumptions they are based. Our aim is to identify significant origins for struggle of students while working with different physical computing devices in order to guide students and teachers to overcome the problems.

We did a pilot-study to get a list of exemplary issues, students struggle with while working with while working with physical computing devices. In order to systematize the issues, we defined categories of origins. To locate the issue origins, we are developed a model of how information is processed in different stadiums in a a physical computing device. Based on that model, teacher-interventions can be investigated.

2 OBSERVATION OF DIFFERENT STUDENT ISSUES DURING A PILOT STUDY

2.1 Pilot-Study in Context of a Computer Science Competition

In order to get an overview of different issues, students struggle with, we did a pilot-study in the context of a computer science competition our work group is

[a] https://orcid.org/0009-0009-7866-1128
[b] https://orcid.org/0009-0002-2942-765X
[c] https://orcid.org/0000-0002-9411-920X

Schätz, E., Hellmig, L. and Martens, A.
Analysis of Student-Problems While Working with Physical Computing Devices.
DOI: 10.5220/0012590000003693
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Computer Supported Education (CSEDU 2024) - Volume 1*, pages 322-329
ISBN: 978-989-758-697-2; ISSN: 2184-5026
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

involved to. This competition (its name is *Landesolympiade Informatik*, short: *LOI*) consisted out of five preparation- and one competition day. About 45 students coming from different schools from 7th to 9th grade participated in this event. Within this competition, the students worked with robot cars as shown in Figure 1 which have been designed and built by our work group.

During the five preparation days, we taught the basics about the usage of the robots in order to balance different backgrounds of the students coming from different school with a diverse quality of computer science education. At each preparation day, there was always one employee from our work group accompanied by two to four university students.

For this work, we concentrate on the observations we made during the preparation days. During those days, we typically gave learning tasks to the students, combined with frontal teaching blocks. Most of the students were working in pairs. If they struggled at a task, they called for help by one of us.

2.2 Physical Computing Device for a Computer Science Competition

In preperation of this competition, we designed a physical computing device and built about 30 exemplars. We called it *Floid* which stands for **f**ancy-**l**andes**o**lympiade-**i**nformatik-**d**evice.

Floid is a simple robot car which can be programmed with the block based development environment Microsoft MakeCode. We provide a library (`github.com/eschaetz/loi-mv-sek1`) for it, which contains blocks for all functions and imports the necessary drivers. The robot uses the BBC Micro:Bit V1 (Brandhofer and Kastner-Hauler, 2020) as computing device. Floid has a number of actuators:

- A differential-drive (Haun, 2013) consisting out of four DC-motors; two on each side. The motors on each side are wired parallel, which means they can not be controlled independently. The robot steers, if the motors at one side rotate faster or in a different directions than the motors at the other side.
  The in the library included block `drive` contains two parameters: steering from -10 to 10 (left to right) and power from -10 to 10 (full power backwards to full power frontwards)
- One NeoPixel, which is a ring of eight RGB-LED at the bottom of the robot
- One display on the top of the robot which can display 2x16 characters

On the sensor-side it has :

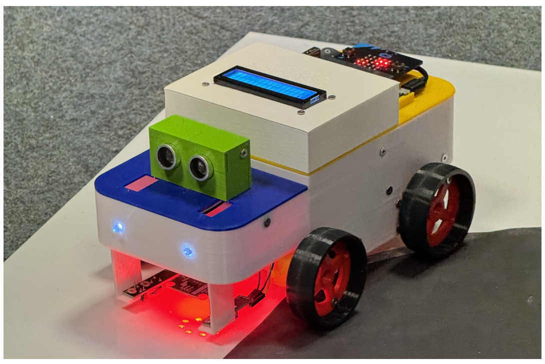

Figure 1: Floid, a simple robots built by out workgroup, students were working with during our pilot-study.

- One ultrasonic distance sensor (Type SR04)
- Two digital line-tracking-sensors which return only 1 for a dark and 0 for a bright surface

All components are shown in Figure 2. Along those robot-specific components, students can use the sensors which are built-in to the Micro:Bit, such as radio, an ambient-light sensor, an accelerometer and two buttons (Brandhofer and Kastner-Hauler, 2020)(Microbit Foundation, 2020)(Hüwe and Hüwe, 2019).
The robot is powered by a power-bank which is in the bottom of the robot. The chassis is completely 3D-printed. This way, the robot looks really unique and we can easily reproduce and replace broken parts or develop new extensions for it.
All in all this robot is quite susceptible for defects, which are typically easy to repair. In the case of our study, this can be seen as an advantage, since that gave us the opportunity to observe how students realize defects and how they handle them.

2.3 Observed Student-Statements

Students typically described error symptoms to us but called it *error*. Those symptoms looked quite diverse but can be summarized in the following seven most significant observed-statements (OS).

- **OS1.** Robot doesn't start moving
- **OS2.** Motors rotate with different speeds, even though the set speed did not change (this is especially a problem in turns, as a 90° turn can take 1000 ms up to 1500 ms or even more)
- **OS3.** Data shown on the display is not correct
- **OS4.** Line-Tracking-Sensors don't recognize a line
- **OS5.** Values from distance are not correct
- **OS6.** Pressed buttons are not recognized
- **OS7.** Received radio signals are not recognized

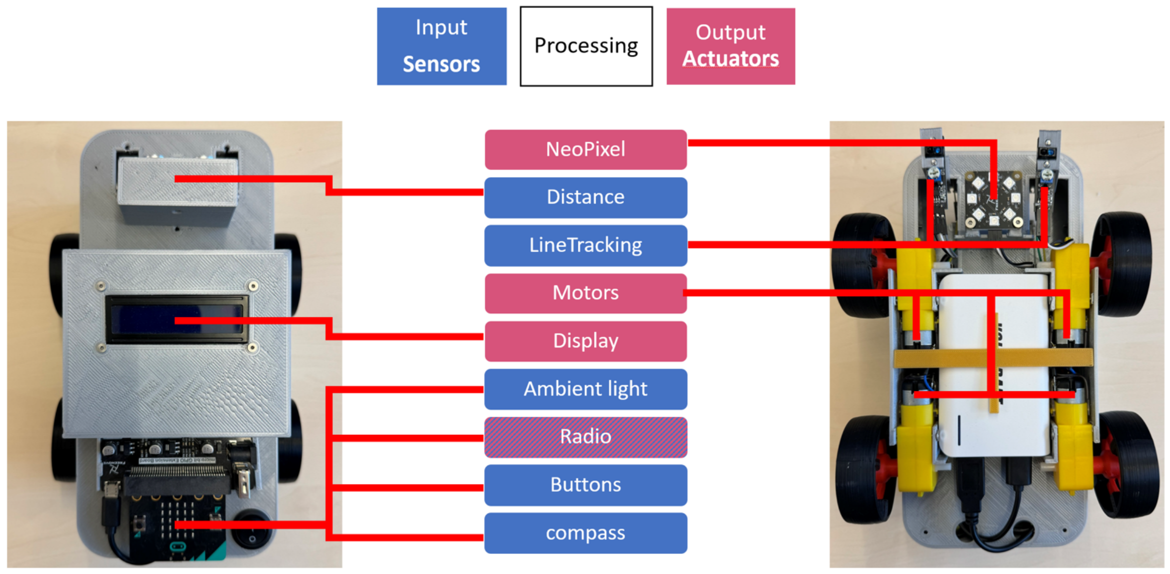

Figure 2: Components of Floid.

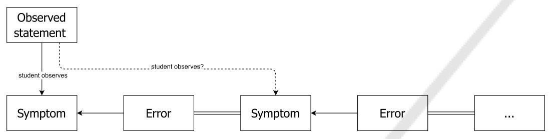

Figure 3: Errors causing symptoms which can be symptoms itself caused by other errors.

3 ISSUE ORIGIN

3.1 Student Statement vs. Symptom vs. Error

Students typically told us they have an *error* but most of the time described an *error symptom*. The errors which caused the symptoms were quite diverse. At this point it needs to be said, an error, which causes a symptom, can sometimes be a symptom resulting of another (deeper) error. Referring to Figure 3, different students described symptoms at different stages. In order to fix the students issue, it is necessary to identify the initial error.

3.2 Different Categories of Origins

In many cases, the symptoms were caused by typical programming errors which were more or less hard to find while looking at the code. In order to identify and solve those errors, students use different debug strategies as described by Romeike and Michaeli (Michaeli and Romeike, 2019). However, those strategies mostly are meant for classical software development at a PC and do not focus on physical computing.

Sometimes, symptoms have been caused by hardware defects for example due a broken motor, a faulty sensor or a loose wire. These errors can appear on classroom PCs as well, but this usually happens not as often as with physical computing devices since personal computers usually include hardware health checks and are usually more robust than physical computing devices like our robots. We built a test routine which helped us to identify broken parts at the robot.

In some cases, the observed symptom had its origin in neither one of those two categories, as the hardware was working all in all appropriate and the software seemed to be correct as well but the program still not worked as it was supposed to. Therefore, there are errors existing, which are neither resulting from broken hardware, nor from software bugs, students made. This leads us to define a third error category: Sometimes error-symptoms are caused by hardware specific reasons, meaning hardware is used in an inappropriate way by disregarding hardware specifications.

In conclusion the observed errors can be divided into three categories:

1. **Category A.** ***Student Caused Bug***
   Those errors are caused by the students.
2. **Category B.** ***Hardware Specification Errors***
   Errors, which appear when hardware is used in an inappropriate way.
3. **Category C.** ***Hardware Defect***
   Errors which are caused by broken components

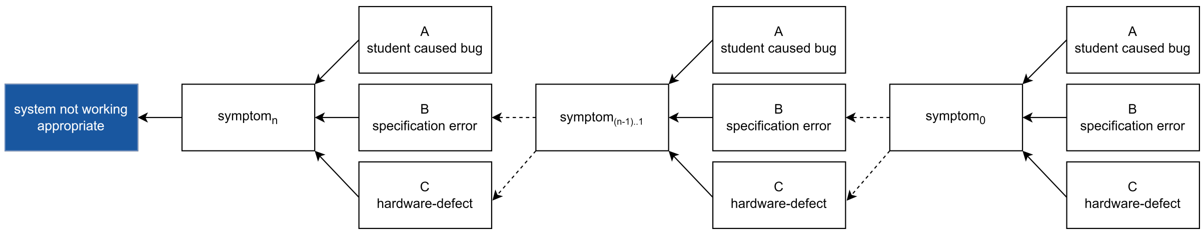

Figure 4: Errors from one category causing errors from another one.

Table 1: Examples of hidden errors from different origin categories.

	Hidden error (exemplary)	**Origin**
OS1	Program is stuck in an infinite loop	A
OS1	Motor is not given enough power	B
OS1	Motor is broken	C
OS3	Wrong variable is shown on display	A
OS3	Two threads are trying to access the display simultaneously which leads to interfering data on the I²C Bus	B
OS3	Loose wire	C
OS5	Measurement is set to inches instead of centimeters	A
OS5	Sensor values are oscillating and need to be smoothed	B
OS5	Sensor is physically broken	C

As shown in Figure 4, errors from one category can cause errors from another one.

The examples stated in Table 1 show, a symptom can be caused by several errors coming from different categories. This increases the complexity of finding errors while working with physical computing devices significantly.

3.3 Borders Between Those Categories

For some examples shown in 1, the category are quite clear and easy to determine. However, in some cases the border between two categories can be fluent. If a hardware component is not used in an appropriate way, this can also be seen as a software bug. However, from the view of a student, the software is correct.

The border between categories A and B can be defined by using a simulation software of the robot: If the program is working in the simulation but not on the real device, than the error is from category B or C, if it does not work in the simulation, than it is in category A. Off course this border depends on which special cases are included in the used model of the simulation software. This way, the border can be defined starting from category A.

The border between B and C can be defined by hardware tests: If the hardware passes the test, than the error is from A or B, if it fails, than it is from C. This procedure requires hardware-tests which detect every kind of hardware anomaly. This way the border is defined outgoing from category C.

In conclusion category B combines all errors which are in between those two borders and can only be defined from the outside, nearing from both neighbor categories. This leads to the insight, that category B is an abstract category: In case a provided simulation software to the robot is complete and a provided hardware test detects every hardware anomaly, every error would either be in A or C. However, a complete simulation is almost impossible to create, since the underlying model would need to include every single behavior of the device and every relevant physical law.

The approach of defining the border between A and B via a simulation has its weak point as there is not a simulation software available for every physical computing device. In this case, a simulation can be replaced by assumptions students have about the robot: As every simulation application is based on a model of the device, a student is creating a program assuming different behavior patterns of the device. Meaning every student has its own model of the robot in its head. This leads to the insight, that the border between A and B can be individual for each student: If a student assumes a motor is rotating once it gets electricity than the error would be in B, if the provided power is not enough for the motor to turn. If a student is aware, that there is a minimum of power a motor needs, than the problem is in A for this particular student.

Analog to a disappearing category B if a complete simulation exists, there would be no category B if a student understands every particular part of a device and has no black-box left. This scenario is as unrealistic as a complete simulation.

Concluding, the more complex a device is, the more it is unrealistic to create a complete model of it, either in context of a simulation or in a students head.

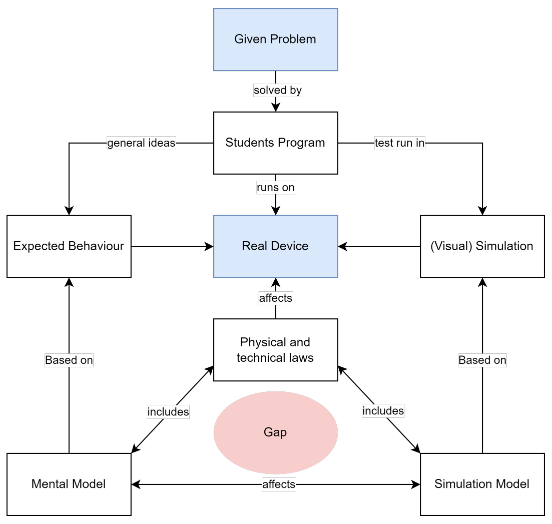

Figure 5: Origin category B can be seen as the gap between students' mental models of the device or a simulation model and the physical and technical laws.

This leads to the point of an always existing category B with fluent borders. It can been interpreted as the gap between the physical and technical laws and the students' mental model of the robot or the simulation model as shown in Figure 5.

We have observed that students tend to assume the problem causing their symptom is located in A or C. If they were not able to find an error in A they frequently claimed their robot is broken. Not only a large number of students had problems identifying errors in B, many students also did not realize the existence of a category B.

4 DATA HANDLING IN PHYSICAL COMPUTING DEVICES

As physical computing stands for the student activity of developing computer systems which interact with their environment via sensors and actuators (Przybylla and Romeike, 2017), errors can appear not only in the procession of data but also in the generation of sensor values or during the action performance of an actuator. Thus, in contrast to pure virtual programming (e. g. a console program or scratch), student's have to care not only about the *P* of the IPO-model but also on the *I* and *O*. Therefore a closer look on this parts in necessary.

4.1 Sensor Data Handling

Sensors are often seen as black-boxes, especially by students. However, in order to understand and solve specification errors, it may be helpful to have some knowledge about their basic functionality. Thus, we are proposing an abstract model of six stadiums of sensor data.

The model is outgoing from a ***given problem*** **(stadium I0)** which can be named explicitly in a task or can appear implicitly while solving another problem. In order to solve the given problem, a physical computing device is monitoring a ***technical observable natural behavior*** **(stadium I1)** using sensors which determines a value on a basis of physical laws. As a result, a sensor provides ***analogous signals*** **(stadium I2)** with a continuous domain. Those signals need to be converted into ***digital values*** **(stadium I3)** on a discrete value margin. While the digital values represent the output of the sensor, the data needs to be converted into a ***common measurement*** **(stadium I4)** in order to be comparable to requirements in the given tasks or data from other sensors. Depending on the functionality of the sensor, the data can be of different quality. Due mathematical smoothing algorithms the raw values can be converted into ***verified values*** **(stadium I5)**.

The steps to get information from one stadium to another address various STEM-disciplines, as illustrated in Figure 6.

Different devices and components return the value at different states to the student: The ambient-light sensor of the BBC Micro:Bit for example returns the values after stadium I3 (Microbit Foundation, 2020), while distance values of the ultrasonic sensor SR04 are returned in centimeters which is stadium I4. Thus, I4 and I5 are optional stadiums as illustrated in Figure 6.

4.2 Proposal of an Extended IPO-Model

The proposed model in Figure 6 addresses only the process of generating sensor-data. However, errors may also appear during the performance of an actuator. Therefore, the model needs to be extended by the output components.

The program of the students affects the processing part of the IPO-Model and controls the output components, in this case the actuators.

In our model, actuators get data either in form of a ***common measurement*** **(stadium O0)** or as ***digital data*** **(stadium O1)**. This data gets converted into ***analog signals affecting the actuating components*** **(stadium O2)**. Therefore, the ***actuator is performing***

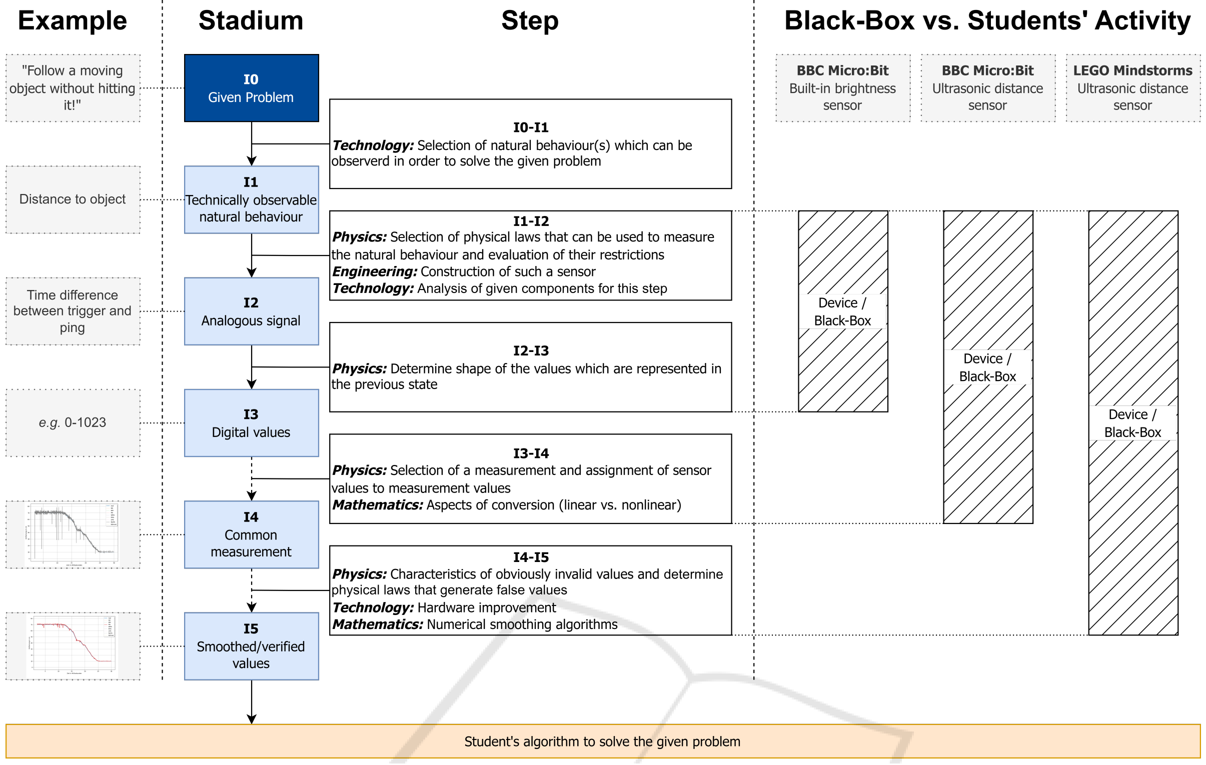
Example
Stadium
Step
Black-Box vs. Students' Activity
"Follow a moving object without hitting it!"
I0 Given Problem
I0-I1
Technology: Selection of natural behaviour(s) which can be observerd in order to solve the given problem
BBC Micro:Bit Built-in brightness sensor
BBC Micro:Bit Ultrasonic distance sensor
LEGO Mindstorms Ultrasonic distance sensor
Distance to object
I1 Technically observable natural behaviour
I1-I2
Physics: Selection of physical laws that can be used to measure the natural behaviour and evaluation of their restrictions
Engineering: Construction of such a sensor
Technology: Analysis of given components for this step
Time difference between trigger and ping
I2 Analogous signal
I2-I3
Physics: Determine shape of the values which are represented in the previous state
e.g. 0-1023
I3 Digital values
I3-I4
Physics: Selection of a measurement and assignment of sensor values to measurement values
Mathematics: Aspects of conversion (linear vs. nonlinear)
I4 Common measurement
I4-I5
Physics: Characteristics of obviously invalid values and determine physical laws that generate false values
Technology: Hardware improvement
Mathematics: Numerical smoothing algorithms
I5 Smoothed/verified values
Device / Black-Box
Device / Black-Box
Device / Black-Box
Student's algorithm to solve the given problem

Figure 6: Process of generating data via sensors though different stadiums.

the action **(stadium O3)**. In some cases, the ***problem is solved*** **(stadium O5)** at this point, in other cases a ***verification*** **(stadium O4)** takes place. This can either be done by a student which would lead to state I1, meaning that sensors need to verify the action of the actuator or by ***hardware implemented self correcting routines*** **(stadium O4b)**.

As the model now includes the whole journey, data goes through a physical computing system, including input and output components, it can be seen as an extended or more detailed IPO-Model.
As shown in Figure 7, this process is not always linear, as several states are optional and data can take a loop through the model. Different devices, such as Lego Mindstorms (LEGO Group, 2022), OzoBot (Ozo Edu Inc., 2022), BBC Micro:Bit (Microbit Foundation, 2020), Calliope Mini (Calliope, 2022), Sphero Bolt (Sphero, 2022) are very diverse and return data at different stadiums to the student. Thus, students have to do some steps themselves in order to work with data in different stadiums, while some parts are hidden in black-boxes as shown in Figure 6.

4.3 Error Focuses Within Extended IPO-Model

While students run through the Physical Computing Process proposed by Schulz and Pinkwart (Schulz and Pinkwart, 2016) they have to think about different connections between the stadium of our model at different places during this process. For example when they are at the point of choosing input and output they have to think about how to come to stadiums I1 and O3. Depending on the chosen device, they don't need to care about some steps and stadiums at this point.

However, while "identify the problem" in "evaluation" of the Physical Computing Process (Schulz and Pinkwart, 2016), we have observed during our prestudy they have to think about all stadiums, as symptoms can be caused by errors from each category. Errors usually appear during the steps between different stadiums. Certain errors sometimes can only be from certain origin categories as shown in Figure 7.

5 DISCUSSION

Our model may help to differ errors from different origins. Due different strategies, the amount of errors

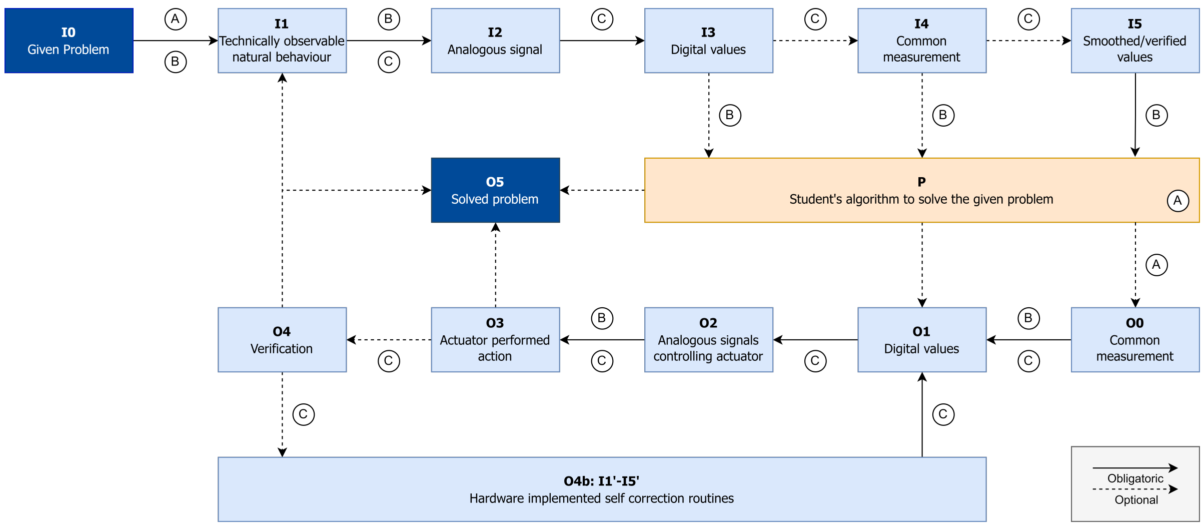

Figure 7: Extended IPO-Model according to data in physical computing applications.

can be in- or decreased, for example can the development environment help to reduce errors from category A and until a certain point also from category B. The usage of a more robust device may reduces the number errors from category C. However, a system is never going to be complete and errors are always possible. From a pedagogical perspective, this insight leads to the question about the pedagogical potential of errors from different categories. It needs to be deliberated between the frustration students generate if a device doesn't work as supposed to and they don't find the error in their program and how problems from different categories force students to develop and practice strategies to ensure the problem is neither from B or C; and if so, how to locate it. It needs to be determined, whether this helps reaching standards of computer science education as for example Bildungsstandards Informatik (Röhner, 2016) or if the fears of the teachers of our workshop were correct and those errors overcharge students and teachers as well.

6 CONCLUSION

In this work, we have designed a pilot-study in order to find out exemplary issues, students struggle with, while working with physical computing devices. For this study, we have constructed out own physical computing device *Floid* in appereance of a robot car. This device is unique and generates several issues, which can be seen as an advantage for our scientific purpose – a more robust device may lead to less issues, but there are most likely be some issues left. Our device gave us the opportunity observing students, how they handeled those issues.

We developed a model of different issue origins and connected the observed issues to those categories. Even though quite some issues were caused by the device, there were also some of them caused by students programs or by disregarding specifications.

In order to locate the issues and errors from different categories, we developed a model of stadiums, information passes in a physical computing device. This model can be seen as an extended or more detailed IPO-model.

7 OUTREACH

As mentioned, the study was still a pilot-study. We are going to repeat it the upcoming fall. as the focus on the pilot-study was evaluating which issues students struggle with, our aim is to evaluate different interventions during the next study.

REFERENCES

Brandhofer, G. and Kastner-Hauler, O. (2020). Der micro:bit und Computational Thinking. *Lehr-Lernprozesse gestalten, analysieren und evaluieren (2020)*, page 20.

Calliope (2022). Calliope.

Haun, M. (2013). *Handbuch Robotik: Programmieren und Einsatz intelligenter Roboter*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hüwe, P. and Hüwe, S. (2019). *IoT at Home: Smart Gadgets mit Arduino, Raspberry Pi, ESP8266 und Calliope entwickeln*. #makers do it. Hanser, München.

LEGO Group (2022). MINDSTORMS EV3 | MINT ab der neunten Klasse.

Michaeli, T. and Romeike, R. (2019). Debuggen im Unterricht – Ein systematisches Vorgehen macht den Unterschied. In *Informatik für alle*. Gesellschaft für Informatik.

Microbit Foundation (2020). The new BBC micro:bit with speaker and microphone.

Ozo Edu Inc. (2022). Ozobot | Robots to code and create with.

Przybylla, M. and Romeike, R. (2017). Von Eingebetteten Systemen zu Physical Computing. In *Informatische Bildung zum Verstehen und Gestalten der digitalen Welt*, pages 257–266, Oldenburg.

Röhner, G. (2016). Bildungsstandards Informatik - Sekundarstufe II. page 88.

Scheglmann, R. (2023). Calli:Bot – unsere-schule.org.

Schmalfeldt, T. and Przybylla, M. (2021). Erhebungsinstrument für Überzeugungen zum Physical Computing. In *Bildung von Lehrkräften in allen Phasen*.

Schulz, S. and Pinkwart, N. (2016). Towards Supporting Scientific Inquiry in Computer Science Education. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*, pages 45–53, Münster Germany. ACM.

Sphero (2022). Coding Robot: Sphero BOLT | Teach STEM for Kids.