ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

THE EFFECT OF VISUAL AND TEXT INTERFACES IN TEACHING ROBOT PROGRAMMING

M.Sc. THESIS

Besim Baransel Bağcı

Department of Computer Engineering

Computer Engineering Programme

DECEMBER 2017

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

THE EFFECT OF VISUAL AND TEXT INTERFACES IN TEACHING ROBOT PROGRAMMING

M.Sc. THESIS

Besim Baransel Bağcı
(504141502)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Asst. Prof. Dr. Gökhan İNCE

DECEMBER 2017

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ROBOT PROGRAMLAMA ÖĞRETİMİNDE GÖRSEL VE METİN ARAYÜZLERİN ETKİSİ

YÜKSEK LİSANS TEZİ

Besim Baransel Bağcı
(504141502)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Asst. Prof. Dr. Gökhan İNCE

ARALIK 2017

Besim Baransel Bağcı, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504141502 successfully defended the thesis entitled "THE EFFECT OF VISUAL AND TEXT INTERFACES IN TEACHING ROBOT PROGRAMMING", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Dr. Gökhan İNCE** Istanbul Technical University	.....
Jury Members :	**Asst. Prof. Dr. Yavuz Samur** Bahçeşehir University	.....
	Assoc. Prof. Dr. Sanem Sarıel Istanbul Technical University	.....
		.....

Date of Submission : **17 November 2017**
Date of Defense : **12 December 2017**

to my family

FOREWORD

In the long run of master's degree, I would like to thank to my advisor Asst. Prof. Dr. Gökhan İnce for being a great advisor and his support in all my desperate times. His guidance and comments about my work encouraged me to work harder and I really appreciate his helpfulness in my study period. I would also like to thank Asst. Prof. Dr. Yavuz Samur and Assoc. Prof. Dr. Sanem Sarıel for their valuable feedback on thesis defense.

For the teaching experiments and test, we have studied with students from İstanbul High School also known as İstanbul Erkek Lisesi, and The Educational Volunteers Foundation of Turkey (TEGV). I would like to thank to them for their support, time and willingness to participate in our experiments. Especially Filiz Erdoğan, manager of the TEGV İpek Kıraç Unit, has been really helpful when we were working with younger students.

Last but not least, I would like to thank my collegues İhsan Halıcı and M. Adem Çelebi for their efforts in teaching and test assessments.

December 2017

Besim Baransel Bağcı
(Computer Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

AESL : Aseba Event Scripting Language
DC : Direct Current
LED : Light Emitting Diode
TAESL : Turkish AESL
VPL : Visual Programming Language

LIST OF TABLES

Page

LIST OF FIGURES

Page

THE EFFECT OF VISUAL AND TEXT INTERFACES IN TEACHING ROBOT PROGRAMMING

SUMMARY

Programming is a popular subject in education and experts emphasize the importance of teaching programming to the children in the high school or even earlier. In European countries, ministries of education give efforts to integrate programming education to school curricula. Some of these countries already have programming classes in high schools and consider to give programming class to younger students. In this respect, methods of the programming education is an important subject and deciding on more successful methods is the problem of this area. In this work, we address this problem and compare different programming interfaces to each other. While teaching programming, we use an educational robot to increase the understanding of the programming environment.

Thymio, which is the educational robot we choose, does not require any knowledge of electronics beforehand and it is in no need of hardware related constructions before the programming. Therefore, when using Thymio our students can focus only the programming part and use different prebuilt functions of the robot. Thymio is simple structured, has a special programming language and development environment for different operating systems and more importantly it is more accessible and suitable for children because it was made of simple and limited number of sensors and actuators. Also Thymio supports two different programming interfaces. First one is text based programming language which is called Aseba Event Scripting Language (AESL) and the other is icon based programming interface which is called Visual Programming Language (VPL).

This study focuses on the programming interfaces and their effects on the performance of programming education. Therefore, we use text programming and visual programming interfaces to teach programming to students. VPL is a purely icon based programming interface and it does not have keywords. Robots can be programmed using only drag and dropping the required icons on the interface. Students do not need to use keyboard and they are not required to know any technical details. On the other hand, AESL is keyword based text programming language which has syntax rules with different keywords. Basic programming features such as variables, decision mechanism, loops etc. are included in the language. It has been designed as event based language to be compatible with robot events. To be able to use AESL interface, students must know basic English and must understand English keywords. To test the effect of this requirement, in addition to original AESL interface we developed an equivalent Turkish keyword based AESL, which we call TAESL. In TAESL, all the keywords and native functions are replaced with Turkish keywords. To be able to do this, we modify the development environment and robot's firmware.

To test our interfaces with the students, we also prepared a curriculum for teaching programming with a robot. We prepare three lessons according to text

programming and visual programming interface structures and Thymio's features. First lesson generally covered introductory subjects to programming and development environment. In the second lesson, we focused on the decision mechanism with minor sensors and actuators of Thymio. In the third lesson, we covered complementary elements to robot components and programming features. In addition to new topics, we have reinforced previous topics to reach a better understanding of algorithms as well as introducing new topics. Our teaching strategy as well as the consequent assessment experiments focused on basic programming subjects. We kept the lectures rather simple, so that we can ignore effects of personal skills and cognitive levels of the students. Also the languages, VPL and AESL, do not need long code portions to control the robot; so we avoid using complex algorithms and long code parts. Almost all of the examples consist of a few lines of code. This minimizes the complexity of teaching, thus we ignore the algorithm creation, debugging and testing effects on our evaluation. In order to reduce the effects of lecturer, we prepare in the form of recorded lectures based on video for each lesson. In addition to that, we organize the lectures as one to one training, thus we minimized the effects of the classroom environments in teaching. Each lecture is planned to be a traditional 45-minutes long session. First 20 minutes of the lesson is devoted to the lecture part. In this section, students watch the lecture videos and ask their questions. Next 10 minutes are reserved for practicing with the robot freely. The final 15 minutes section is dedicated to the exam part. In the evaluation phase, collected codes are tested by our team in predesigned evaluation environment which has obstacles and cliffs at necessary locations. Each objective is assessed in the given order and the student is considered successful only if the code produces the expected outcome.

In our experiments, 32 students, within ages of 7 to 24 with an average 14.3 and standard deviation 5.25 participated. The majority of the students had no experience with programming before and rest of them had barely some experience on programming. We divided students into 3 different groups, students who studied visual programming (Group 1), students who studied text programming (Group 2) and students who studied text programming after visual programming (Group 3). According to the results of experiments, students who studied with visual programming have been more successful than students who studied only text programming. This is an expected result because visual programming is faster and simpler to program. So, it can be said that the visual programming languages are better for learning programming. When we compare the performance of the students who studied only text programming, with the performance of the students who studied visual programming first and then text programming, we observed that students who studied text programming after visual programming have been more successful than students who studied text programming only. This comparison shows that learning programming with visual languages helps to learn text based programming languages later. Another performance criterion in our tests is the time spent on each interface. While time spent on the visual programming interface is always less than the others because of the simplicity of the visual programming, the effects of the visual programming languages can be seen on the time spend for programming on between text programming after visual programming and just text programming interfaces. According to these result we can say that learning programming with visual languages helps to adapt text based programming faster. We also compared the results of the students who study with original text programming to the results of the students who

study with TAESL. In our test results there is no significant difference between the success rates of original AESL and TAESL. Average success rates is nearly identical in original AESL and TAESL.

In conclusion, we used different programming interfaces to teach programming using robots and compared the results of the experiments. According to result of our experiments, visual programming languages are found to be easier and quicker to learn. In the experiments, using visual programming students had higher success rates overall. In addition to that, when learning text programming, if students have visual programming background before, they had more successful results on learning text programming concepts. So, we verified that if visual programming languages are used for learning programming first time, students can adapt to other programming languages easier. Also, according to our experimental results, using a mother-tongue based programming language does not have any effect on the learning of programming.

ROBOT PROGRAMLAMA ÖĞRETİMİNDE GÖRSEL VE METİN ARAYÜZLERİN ETKİSİ

ÖZET

Programlama, eğitim alanında popüler bir konudur ve uzmanlar çocuklara lisede ya da daha erken yaşta programlama öğretmenin önemini vurgulamaktalar. Bir çok Avrupa ülkesinde, eğitim bakanlıkları programlama eğitimini okul müfredatlarına eklemeye çalışmaktadır. Bazı ülkeler liselerde zaten bir programlama sınıfına sahipler ve daha genç öğrencilere programlama sınıfı sağlamayı düşünüyorlar. Bu açıdan, programlama eğitiminde kullanılacak yöntemler çok önemli bir konu olmaktadır ve daha başarılı bir yöntem belirlemek bu alanın en büyük problemlerinden biridir. Bu çalışma, bu soruna değinmekte ve farklı programlama arayüzlerini birbirleriyle karşılaştırmaktadır. Programlamayı öğretirken programlama ortamının anlaşılmasını arttırmak için ise bir eğitim robotu kullanılmıştır.

Seçilen eğitim robotu Thymio, önceden herhangi bir elektronik bilgiye sahip olunmasına ihtiyaç duymamakta ve programlamadan önce donanımla ilgili bir yapılandırmaya ihtiyaç duymamaktadır. Bu nedenle, Thymio kullanırken öğrenciler yalnızca programlama kısmına odaklanarak robotun farklı işlevlerini kullanabilirler. Thymio basit yapılıdır, üzerinde yeterince sayıda sensör barındırır. Kendine özel bir programlama dili ve farklı işletim sistemleri için geliştirme ortamı bulunmaktadır ve daha önemlisi daha erişilebilir ve çocuklar için daha uygundur. çünkü basit ve sınırlı sayıda sensör ve çıkış aygıtı ile yapılmıştır. Ayrıca Thymio iki farklı programlama arabirimini destekler. Birincisi, Aseba Event Scripting Language (AESL) olarak adlandırılan metin tabanlı programlama dili ve diğeri Visual Programming Language (VPL) olarak adlandırılan ikon tabanlı programlama arabirimidir.

Bu çalışma, programlama arayüzleri ve bunların programlama eğitiminin başarısı üzerindeki etkileri üzerine yoğunlaşmıştır. Bu nedenle, öğrencilere programlama öğretmek için AESL ve VPL arayüzleri kullanılmıştır. VPL tamamen simge tabanlı bir programlama arabirimidir ve anahtar kelimeler içermez. Robotlar, arayüzde gerekli simgeleri sürükleyip bırakarak programlanabilir. Öğrencilerin klavyeyi kullanmaları veya herhangi bir teknik ayrıntı bilmeleri gerekmez. Öte yandan, AESL, farklı anahtar kelimelerle sözdizimi kurallarına sahip kelime tabanlı bir metin programlama dilidir. Değişkenler, karar mekanizması, döngüler vb. temel programlama özelliklerini içerir. Robot olaylarıyla uyumlu olması için olay tabanlı bir dil olarak tasarlanmıştır. AESL arayüzünü kullanabilmek için öğrencilerin İngilizce anahtar kelimeleri anlayabilmesi ve temel seviyede İngilizce bilmeleri gerekmektedir. Bu gerekliliğin etkisini test etmek için, orijinal AESL arayüzlerine ek olarak, TAESL denilen eşdeğer bir Türkçe anahtar kelime tabanlı AESL arayüzü geliştirilmiştir. TAESL'de tüm anahtar kelimeler ve yerli işlevler, Türkçe anahtar kelimelerle yeniden adlandırılmıştır. Bunu yapabilmek için geliştirme ortamını ve robot gömülü yazılımı değiştirilmiştir..

Robot ile programlamayı öğreterek arayüzleri öğrencilerle test edebilmek için bir müfredat oluşturuldu. AESL ve VPL arayüz yapılarına ve Thymio'nun özelliklerine

göre üç ders hazırlandı. Birinci ders genellikle giriş konuları ile programlama ve geliştirme ortamına yönelik hazırlandı. İkinci derste, Thymio'nin basit sensörleri ve çıkış aygıtları ile karar mekanizmasına odaklandık. Üçüncü derste, robot bileşenlerinin ve programlama özelliklerinin tamamlayıcı unsurları incelendi. Yeni konulara ek olarak, algoritmaların ve yeni konuların daha iyi anlaşılmasını sağlamak için eski konular pekiştirildi. Öğretim stratejisi ve sonuçta yapılan değerlendirme deneyleri temel programlama konularına odaklandı. Öğrencilerin kişisel becerilerinin ve bilişsel düzeylerinin etkilerini göz ardı edebilmemiz için konuları oldukça basit tutuldu. Ayrıca, VPL ve AESL dilleri, robotu kontrol etmek için uzun kod parçasına ihtiyaç duymadığından, karmaşık algoritmalardan ve uzun kod parçalardan kaçınıldı. Bu nedenle örneklerin neredeyse tamamı birkaç satırlık koddan oluşmaktadır. Bu, öğretimin karmaşıklığını en aza indirdi ve böylece algoritma oluşturma, hata ayıklama ve değerlendirme üzerindeki etkileri göz ardı edilebildi. Eğitmenin etkilerini azaltmak için her ders için video dersi hazırlandı. Buna ek olarak dersler bire bir eğitim şeklinde organize edildi, böylece sınıf ortamlarının öğretmeyle olan etkileri en aza indirildi. Her dersin klasik tarzda 45 dakikalık bir oturum olması planlandı. Dersin ilk 20 dakikası ders bölümüne ayrılmıştır. Bu bölümde, öğrenciler ders videolarını izlediler ve sorularını sordular. Sonraki 10 dakika, robotla pratik yapmak için ayrılmıştır. Son 15 dakikalık bölüm sınav bölümüne ayrılmıştır. Değerlendirme aşamasında ise toplanan kodlar, gerekli yerlerde engeller ve boşluklar olan önceden tasarlanmış değerlendirme ortamında test edilmiştir. Her bir hedef, verilen sırayla değerlendirilmiş ve yalnızca, kodun beklenen sonuca ulaşması durumunda başarılı sayılmıştır.

Testlerimize yaşları 7 ile 24 arasında ortalama 14.3 ve standart sapması 5.25 olan 32 öğrenci katılmıştır. Öğrencilerin çoğunluğu daha önce hiç programlama tecrübesine sahip değildiler ve geri kalanlarının programlama konusunda az seviyede ön bilgileri mevcuttu. Öğrencileri, görsel programlama (Grup 1), metin programlama (Grup 2) ve görsel programlamadan sonra metin programlama çalışan (Grup 3) öğrenciler şeklinde üç farklı gruba ayırdık. Deney sonucuna göre, görsel programlama ile eğitim gören öğrenciler yalnızca metin programlamayı okuyan öğrencilere göre daha başarılı olmuşlardır. Bu beklenen bir sonuçtur, çünkü görsel programlama çok daha hızlı ve basittir. Yani, görsel programlama dillerinin programlamayı öğrenmek için daha uygun olduğu söylenebilir. Yalnızca metin programlamayı öğrenen öğrencileri önce görsel programlama sonra metin programlama öğrenen öğrencilerle karşılaştırdığımızda önce görsel programlama sonra metin programlama öğrenen öğrencilerin yalnızca metin programlama öğrenenlerden daha başarılı olduklarını gördük. Bu karşılaştırmalar görsel dillerle kodlama öğrenmenin daha sonra metin tabanlı programlama dilleri öğrenmeye yardımcı olduğunu gösteriyor. Testlerimizde bir diğer değerlendirme kriteri, her arabirim için harcanan zamandı. görsel programlama arayüzü üzerinde harcanan zaman görsel programlamanın sadeliğinden ötürü daima diğerlerinden daha küçük olsa da, görsel programlama dillerinin etkileri, görsel programlamadan sonra metin programlama ile sadece metin programlama arayüzleri arasında programlama için harcanan zaman farkları kullanılarak görülebilir. Bu sonuçlara göre, görsel dillerle kodlama öğrenmenin metin tabanlı programlamayı daha hızlı öğretmeyi sağladığını söyleyebiliriz. Ayrıca orijinal AESL ile öğrenen öğrencilerin sonuçları ve Türkçe AESL ile çalışan öğrencilerin sonuçları karşılaştırılmıştır. Test sonuçlarımızda orijinal AESL ve Türkçe AESL başarı oranları arasında anlamlı bir fark bulunamamıştır. Orijinal AESL ve Türkçe AESL arasında ortalama başarı oranları neredeyse aynıdır.

Sonuç olarak robotlarla programlama öğretmek için farklı programlama arabirimleri kullandık ve deneylerin sonuçlarını karşılaştırdık. Deney sonuçlarımıza göre, görsel programlama dillerinin daha kolay kullanılabilir olduklarını ve programlamanın daha çabuk öğrenildiğini doğruladık. Deneylerde, görsel programlama ile kodlama yapan öğrenciler genel olarak daha yüksek başarı oranına sahipti. Buna ek olarak, metin programlamayı öğrenirken, öğrencilerin daha önce görsel programlama geçmişlerine sahip olmaları durumunda, metin programlama kavramlarını öğrenmede daha başarılı sonuçlar elde ettiklerini gördük. Görsel programlama dilleri ilk kez programlama öğrenmek için kullanılıyorsa öğrenciler diğer programlama dillerine daha kolay uyum sağlarlar. Ayrıca, testimize göre, bir anadil tabanlı programlama dili kullanmak, programlama öğrenimine herhangi bir etki yapmamaktadır.

1. INTRODUCTION

1.1 Purpose of Thesis

Programming education has long been a controversial issue. For nearly 40 years, effects of the learning programming and the methods of teaching programming languages have been research topics for researchers. At first, programming was being used in electronic device design by adults and most of the early programming languages and interfaces focused on the specific devices and its features. With the technologic development and it's integration to our lives today, programming is in the interest of different disciplines aiming to the audience of much younger students. Nowadays, programming and algorithm knowledge is considered as an essential part of the technology and computer education. Even during the preparation of this thesis programming education in pre-college class is a hot topic, there are different approaches in either curricula or programming environments to teach programming to young students.

Purpose of this thesis is to examine the effects of visual and text-based programming interfaces in the programming education and to help to analyze the effectiveness of different programming interfaces for first programming education. When teaching programming, we used robots in our courses and conducted teaching experiments. The teaching experiments on the different interfaces created objective test results for the assessment of success rates of different programming interfaces. We used this teaching experiments and test results to propose a teaching method for programming. We also investigated the effect of the language in text programming interfaces.

1.2 Literature Review

In the early days of computers, programming languages are considered as an adjunct to other subjects in teaching, especially for mathematics. The first popular example of this approach is LOGO project, which is a programming language specifically designed for

teaching mathematics to children [1]. With the popularization of personal computers, programming languages became a major subject in education. Most of the researches focused on effects of learning programming languages on children's cognitive style, metacognitive abilities and cognitive development [2] [3] [4]. More recent studies suggest that teaching programming languages increase students' problem solving abilities, natural language skills, creativity and ability of working in multidisciplinary subjects [5]. Because of these benefits, teaching programming languages became popular in modern education.

Nowadays, teaching computer languages has been considered as an essential part of the basic sciences and it is recommended that computer languages should be included in the secondary school curriculum or even in primary school curriculum. When teaching computer languages in primary or secondary schools, it is important to note that the main aim is to help to development of students' abilities; not to make students programming experts in early ages which is impossible even for college students in four years of college education [6]. According to a report from European Schoolnet, which is a network organization of 30 European Ministries of Education, computer languages and programming is already part of the curriculum in 12 countries which includes Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Greece, Ireland, Italy, Lithuania, Poland, Portugal and the UK. Also 7 countries which includes Belgium Flanders, Spain, Finland, France, Luxembourg, the Netherlands and Turkey are working on to integrate computer programming to their curricula [7].

When we consider all these developments, it is not arguable anymore whether programming should be part of the curriculum or not. The main focus is on the methods of teaching computer programming. According to a survey of literature, significant researches on teaching computer programming are focused on either curriculum, pedagogy, computer languages or programming tools [8].

In curriculum, researchers put their efforts mostly on to improve teaching methods and concepts. Some of the studies propose new approaches for teaching programming changing the methods and paradigms [9] [10] [11]. Instead of designing a new approach, some of the papers focus on the students' motivation and give an idea about the issues of audience and motivation [12]. In addition to new proposes, some studies focus on the past changes on the curriculum and their effects [13].

On the pedagogical side of teaching, the study subjects vary. Some of the studies give insight thoughts on the cognitive and psychological aspect of programming [14] [15]. Others focus on the effects of teachers, students, environment and the roles and encounters between teachers and students while teaching programming; such as effects of students' approaches to learning [16].

Programming tools generally used for visualization of the programming concepts or in automated assessments of the code. While visualization tools proposed for mostly object or data structure visualization, it is also used for algorithm visualization and visual code editing [17] [18] [19]. Another focus point for tools is the automated assessment tools [20] [21].

For programming languages, there are a lot of discussions on which one is the most appropriate to expose the students as their first programming language. While popular languages such as C, C++, Java and Python are generally used for learning programming in college level, there exist a lot of languages for learning programming [22] [23] [24]. Moreover, some researchers proposed mother-tongue based programming languages, which have keywords in native languages or in native sentence structure; but these proposed languages had limited impact on the general teaching of programming languages [25].

When teaching programming languages started to focus on younger students, proposed languages and interfaces began to shape around younger minds. To teach programming languages to younger children, visual programming languages were created. These visual languages have more flexible syntax and are mostly based on the graphical icons and their relations. Scratch is a popular example for visual programming languages and it is designed for children ages between 8 and 16 to teach programming while they working on more complex and creative projects [26]. Scratch generally is used in studies for giving an introduction to programming concepts without bothering mathematics or syntax details and in some studies used scratch to enhance the programming experience [27] [28] [29] [30]. Also, Scala language along with the Kojo environment is used in studies to give a visual introduction to programming [31] [32]. Studies in this area, generally focused on the user experience of the programming but they do not give an insight to success of the learning programming objectively.

Another approach for teaching programming languages to children emerged from the field of robotics. Instead of focusing on languages and programs on the computer, robots have been used as a tool in teaching programming. Studies which using robots generally focus on younger children, thus in addition to teaching programming, students improve logical thinking abilities [33] [34] [35]. These studies also suggest that the using robots helps to give students a better understanding of programming. By programming robots, constructivist approach of teaching has been used as a more actively in teaching programming languages because of using robots support the basic tenets of constructionism by providing more active learning to students, manipulating robots to think with, providing more powerful and significant projects and giving more change to selfreflection [36] [37].

1.3 Research Questions

In this study, we focused on the effects of the programming languages and interfaces in teaching programming with educational robots. We used robots as programming objects because they provide more tangible programming experience while learning programming and we studied with precollege students to college students when teaching programming. While there exist different approaches to teach programming, other studies don't focus on the effects of different programming interfaces of robots.

This study aimed to clarify the benefits of the visual programming interfaces as either main programming interfaces or an introduction interface before others. In this study we will give effort to answer four research questions to clarify these benefits.

- **RQ1) Are visual interfaces more effective compared to text-based interface in learning programming languages?** To answer that question, we will compare the success rates of text programming interfaces and visual programming interfaces on the student groups.
- **RQ2) Do the students learn text programming easier if they learn visual programming first?** To answer that question, we will compare the text programming performance of students who learn by only text programming with the students who learn visual programming before text programming.

- **RQ3) Does the age of students have any effect in learning programming with robots?** To answer that question, we will investigate the success of the students according to their age on different programming interfaces.
- **RQ4) Does the language of the keywords in text programming have any effect in learning programming?** To answer that question, we will teach students English keyword based and Turkish keyword based languages and compare their performance.

2. SYSTEM ARCHITECTURE

Our system environment consist of two different units: an educational robot and programming interfaces to teach programming. When teaching programming, we used a computer in order to run programming interfaces and to connect robot.

In lectures, students watched the lecture videos on the computer and according to tutorials they tried to program the educational robot. Our programming interfaces are installed on the computer and required a cable connection to robot while programming in order to read sensor data and compile the program. When students using the programming interfaces, robot stays connected to computer via cable and charged. Thus, students were be able to watch the sensor data on the computer. After completion of the code, students saved and loaded their code to robot via cable. After that students were be able to disconnect robot from computer and test their program. The educational robot and students' codes are tested on the large table and if they needed, students used the different boxes in order to create an obstacle or a maze. Example figure of the environment shown in Fig. 2.1. The programs written by the students are collected at the end of the each lesson and test on the same environment.

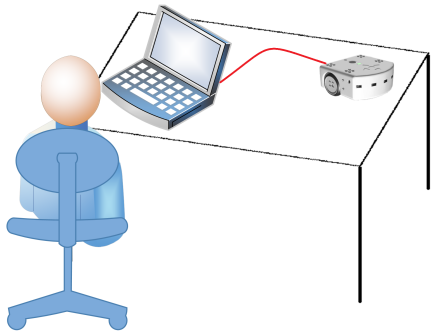

Figure 2.1 : System environment for learning computer programming.

2.1 Educational Robot Platform

When we designed our teaching and test environment, we evaluated different robotic platforms for educational usage such as Lego Mindstorms, Arduino and Thymio [38]

Figure 2.2 : Educational platforms, left Lego Mindstorms, middle Arduino board, right Thymio.

[39] [40]. While Lego Mindstorms is a popular platform and has a lot of functionalities, it requires a hardware mounting, which is building robot using the necessary sensors, motors and other extensions. Arduino is electronic developing board and it supports more complex hardware and software structures, but it requires significant knowledge of electronics before the programming. When comparing other platforms with Thymio, Thymio is simpler and better suited to children. It does not require any knowledge of electronics beforehand and it is in no need hardware related constructions before the programming. All platforms shown in Fig. 2.2. Therefore, we decided to select Thymio because it is simple structured, has enough number of sensors on it and has a special programming language and development environment for different operating systems and more importantly it is more accessible and suitable for children because it is made of simple and with limited number of sensors and actuators [41].

Thymio has 9 infrared (IR) proximity sensors (5 on the front, 2 on the back and 2 on the bottom), 5 capacitive touch buttons (on the top), 1 three-axis accelerometer, 1 thermometer, 1 microphone (for recording or detecting noise level), 1 IR receiver (for remote control or communication), 39 LEDs (front, back, top and bottom), 2 DC motors (which have speed control and speed sensors) and 1 loudspeaker [42]. Therefore, it can be used for different tasks such as path tracking, following, escaping, balancing etc. Thymio was shown in Fig. 2.3

In our work, different robot features are used for teaching programming controls. Microphone and touch button sensors are used in lectures for understanding sensor data inputs and taking actions according to them. For the actions, we used LEDs to change color of the robot and wheels to move robot in the test environment. While the robot is moving and microphone sensor is in use, some problems occurred because of the sound of the motors and wheels. When sound sensing is too sensitive, the robot senses

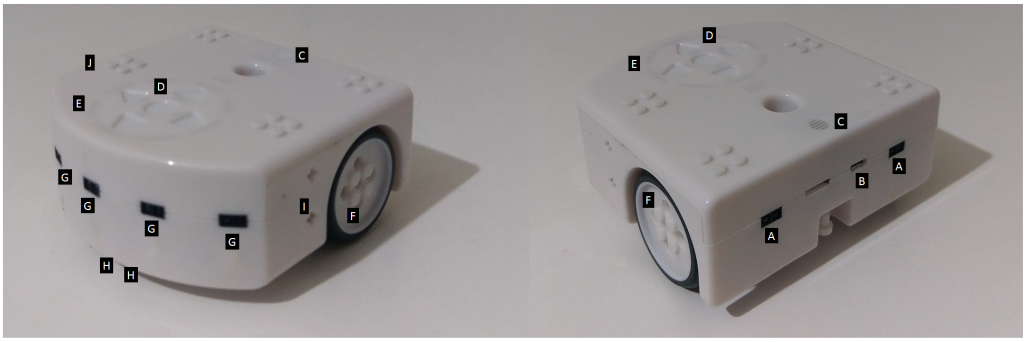

Figure 2.3 : The Thymio robot and components. A) Rear proximity sensors B) USB connection port C) Loudspeaker D) Capacitive touch buttons E) Status leds F) Wheels and speed sensors G) Front proximity sensor H) Ground sensors I) Temperature sensor J) 3-axis accelerometer .

the noise of the motors and its own movement and takes action to this input. To avoid this problem, we adjusted the default sensor sensitivity in our interfaces according to environment and we adapted our tests at the end of the lecture for further isolation. While introducing decision making structures in the robot, we used proximity sensors in the front and the rear of robot for sensing and escaping from obstacles. Also the ground sensors under the robot are used for understanding the environment. Loudspeaker component is used in lectures as an alert method for the sensor inputs and the states. Similar to previous problem, using microphone and loudspeaker together caused undesired side effects. Thus, we designed our test objectives to avoid this specific situation. Also for sensing the robots own movement and the location, we used speed sensor from wheels and accelerometer. While accelerometer is in use, for preventing the negative acceleration effect on the beginning of the movement, we limited the speed of the wheels and accelerometer threshold to be compatible with each other.

2.2 Programming Interfaces

For programming environment, Thymio takes advantage of the Aseba Tools [43]. It supports 3 different types of programming interfaces which can be summarized as text programming, visual programming and blockly interface, which is a mixture of visual and text programming interfaces [44]. Visual programming interface, which is called VPL, consists of only icons and symbols and is independent from any human language. Text programming language, which is called Aseba Event Scripting Language (AESL), is English keyword based scripting language like many programming languages. Blockly interface is a combination of text and visual blocks, which are prewritten code blocks. In order to observe the effects of text programming and visual programming separately, we did not include the Blockly interface in our work, because Blockly contains both visual and text programming parts. So, in our study we used and compared text programming and visual programming interfaces.

2.2.1 Programming via Text Interface

Thymio is programmed with Aseba Event Scripting Language (AESL) which is a scripting language of a modular, distributed and event-based architecture ASEBA [45] [46]. In text programming interface, AESL consists of the English based keywords and native functions on programming. In AESL, program flows are defined by event definitions and each event block runs simultaneously if the necessary event occurs. Event definitions are made with *onevent* keyword and can be supported by decision blocks with *if* and *when* keywords and if necessary with loops. Actions and other changes are made mostly by assigning appropriate variables and array values.

Text programming interface is shown in Fig. 2.4. Beside the main code area, text interface has code control buttons on the top left field. These buttons are used in order to load the written code to robot or to reset the robot. Under this area, there is a dynamic list of robot variables. These variables are generated by the robot and their values are read simultaneously from robot. Programmer can use these values to observe their running code. Also, while writing the code, these variables can be dragged to code area instead of typing. Below this area, there is a native function list

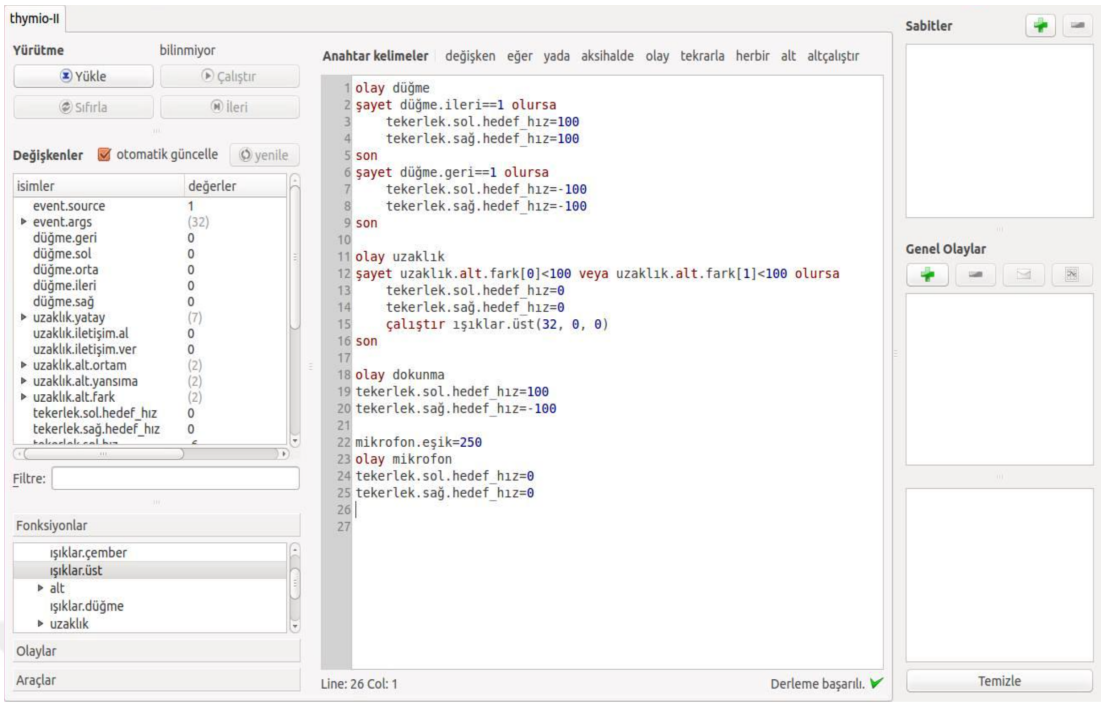

Figure 2.4 : Text Programming Interface.

for the robot. All native functions can be viewed in this area and can be dragged to code area to easy implementation. All event definitions of the robot are listed in the next area. These event definitions are keywords for detecting event type and the robot response to events. Programmer can drag the event keyword from this list to the code area. At the bottom of the left pane, there are robot control buttons. These button can stop the robot and the running code immediately or reset the code while the robot is connected to computer.

Above the main code area, there is hint bar for the most used keywords of the language. These keywords can be dragged to code area easily instead of the typing the code structure. Below the main code area, there is a status bar which shows the code errors or the compilation status. At right side of the screen, there is a pane for user defined constant or events. Programmer can define constant variables or events instead of using predefined ones of robot.

In addition to original AESL, we develop a version of AESL with Turkish based keywords and native functions, TAESL. AESL keywords and their equivalents are shown in Table 2.1. In our tests we use these two versions of AESL to compare effects of the native language in programming learning. The syntax of the TAESL is kept the same with the original AESL but the keywords are changed with Turkish words and all

Table 2.1 : Original AESL keywords and Turkish AESL equivalents.

AESL Keywords	TAESL Keywords
when	şayet
emit	duyur
for	herbir
in	içinaralık
step	adım
while	tekrarla
do	olursa
if	eğer
then	ise
else	aksihalde
elseif	yada
end	son
var	değişken
const	sabit
call	çalıştır
sub	alt
callsub	altçalıştır
onevent	olay
abs	mutlak
return	çık
or	veya
and	ve
not	değil

native functions are replaced with Turkish function names. Example scripts are shown in Table 2.2

2.2.2 Programming via Visual Interface

In visual programming interface, we used Visual Programming Language (VPL) which is developed for Thymio and is suited for children in learning computer languages [40]. VPL does not include any keywords but it only has icons as programming elements. Therefore, there is no language effects on the visual programming interface.

Visual programming interface is shown in Fig. 2.5. Above the main code area, there are main control area which includes buttons. First of these control buttons are related to basic file operations such as opening a new script or saving the current code. At middle of the area, there is a play button which loads the written code to the robot and a stop button which stops the robot and the running code immediately. At left and right side of the main code area there is event pane and action pane respectively. The

Table 2.2 : Original AESL example scripts with Turkish version of AESL.

Original AESL	**Turkish version of AESL**
if *a* == 0 **and** *b* == 2 **then** *a* = 1 **elseif** *a* == 1 **then** *a* = 2 **else** *a* = 3 **end**	**eğer** *a* == 0 **ve** *b* == 2 **ise** *a* = 1 **yada** *a* == 1 ise *a* = 2 **aksihalde** *a* = 3 **son**
when *a* > *b* **do** *leds[0]* = 1 **end**	**eğerki** *a* > *b* **olursa** *ışık[0]* = 1 **son**
while *i* < 10 **do** *i* = *i* + 1 **end**	**tekrarla** *i* < 10 **olursa** *i* = *i* + 1 **son**
for *i* **in** 30:1 **step** -3 **do** *v* = *v* - *i* * *i* **end**	**herbir** *i* **içinaralık** 30:1 **adım** -3 **olursa** *v* = *v* - *i* * *i* **son**

icons on these panes are draggable and are used when creating the code. A status bar is located just above the main code area to show errors or compilation status.

Thymio is event based robot, so when programming the robot with visual interface, each horizontal block represents an event definition on the robot. In main code area, each horizontal block divided to two area by a colon. Left area is reserved for the event definitions and conditions while right side is is designed for the actions. When programmer defining a new feature, events must be dragged from the event pane to left side of the horizontal code block while actions are dragged from the actions pane to the right of the colon. Each code block must have one and only one event definition and at least one action. Multiple action on the horizontal code block start at the same time. For every behaviour of the robot, programmer must implement a new horizontal code block. At the right side if the screen there is a text area which shows the text programming equivalent of the written VPL code.

In the example code of Fig. 2.5, there are 3 horizontal code blocks which represent 3 different events. At first block, first icon represent the forward button event and second icon declare a condition to begin an action. If forward button is pressed, second icon check for the state variables which are also linked on the LEDs on the robot. If all 4

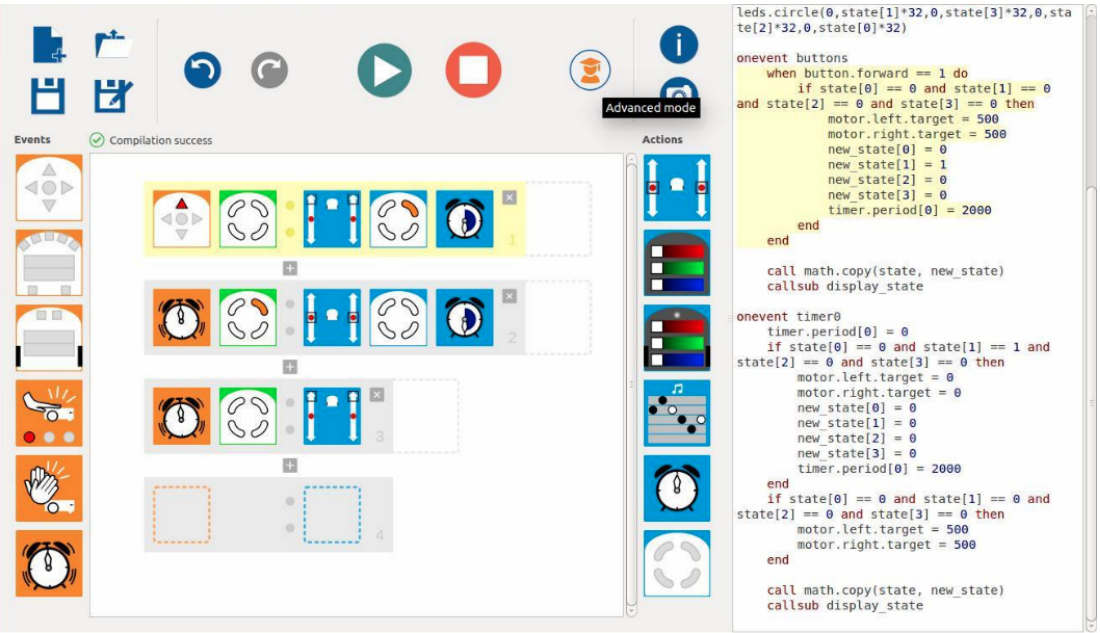

Figure 2.5 : VPL interface and its equivalent AESL code.

state variables are zero, which means all state LEDs are off on the robot, then robot starts the actions. In action side, there are 3 different actions. First icon represent left and right motors. As shown in the icon, both motors set to forward at maximum speed, so robot starts a movement to forward. Second icon represents the state variables and changes the status of robot while enabling one state LED on the robot. Third icon represents a time event trigger. It counts 2 seconds and after that, emits a time event. This trigger is used to create a chain of actions. At the second horizontal code block, there is a time event definition and a state condition. When a time event occured and only top right state variable is enabled, then the action parts starts. The action part quite similar to previous one, except both motors are reset and all state variables are disabled. At the last horizontal block, there are again a time event definition and a condition which requires all state variables are disabled. In the action part, it only starts a forward movement at maximum speed. If the example code are loaded the Thymio, the robot starts a forward movement when forward buttons is touched. After 2 seconds, robots pauses its movement for 2 second, then continues its forward movement forever. In VPL, first two icons in each line represent the conditions and they correspond to *onevent*, *when* and *if* keywords in its AESL equivalent. Other icons in VPL defines the actions and generally represented as variable assignment in AESL.

3. TEACHING AND ASSESSMENT METHODOLOGY

Programming using robots requires both teachers and students to be an active participant of the class. In programming, especially with working long code blocks, minor errors or algorithmic miscalculations make it difficult to identify problem on the code. This leads to a reduction in the amount of time that the teacher allocates to each student in the classroom environment of many students. Also with working robots, each students needs a large workspace to test their codes on the robots. Thus, we adopted the method of individual study in our classroom, with most 2 students per teacher. Also we included pre-recorded video tutorials and tutorials to enhance the clarity and applicability of the subject.

3.1 Teaching Strategy

Our teaching strategy as well as the consequent assessment experiments focused on basic programming subjects. We kept the subjects quite simple, so that we can ignore effects of personal skills and cognitive levels of the students. Also the languages, VPL and AESL, do not need long code blocks to control the robot, thus we avoided using complex algorithms and long code parts. Almost all of the examples are a few lines of code. This minimized the complexity of teaching and thus we ignored the algorithm creation, debugging and testing effects on the evaluation.

For course materials, in order to reduce the effects of lecturer, we prepared recorded videos for each lesson. In addition to that, we organized the lectures as one to one training, thus we minimized the effects of the classroom environments in teaching. During lectures, the students watched training videos and later they asked their questions to the lecturer in a given time.

Each lecture was planned to be a classic 45-minutes long session. First 20 minutes of the lesson is devoted to the lecture part. In this section, students watched the lecture videos and asked their questions. Next 10 minutes are reserved for practicing with the robot freely. The learning part of each session is limited to 30 minutes, because our

video lecture contents are structured to short periods and during that time students have time for thinking and interaction with the lecturer. Also by keeping the lesson period short, we tried to keep the students focused. The final 15 minutes of each section is dedicated to the exam part.

3.2 Lecture Content

We prepared three lessons [1] according to AESL and VPL interface structures and taking into account Thymio's features. First lesson generally covered introduction subjects to programming and development environment. Also in first lesson, a brief description of Thymio and its features are also included. In the second lesson, we focused on the decision mechanism with minor sensors and actuators of Thymio. In the third lesson, we covered complementary elements to robot components and programming features. In addition to new topics, we have reinforced previous topics to reach a better understanding of algorithms. Lecture topics shown in Table 3.1, Table 3.2 and Table 3.3.

Table 3.1 : Course topics of Lecture 1.

Lecture 1
1.1) Introduction to Thymio and its components
1.2) Introduction to development environment and interface
1.3) Event definition and event listening
1.4) Usage of the top-buttons
1.5) Usage of the touch and sound sensors
1.6) Usage of the wheels
1.7) LED lights and basic data operations

Table 3.2 : Course topics of Lecture 2.

Lecture 2
2.1) Usage of the front proximity sensors for obstacle detection
2.2) Usage of the rear proximity sensors for escaping
2.3) Usage of the ground sensors to avoid falling off the cliff
2.4) Decision mechanism with *if* and *when* keywords
2.5) Usage of loudspeaker output for sound alert

[1] Lectures can be accessed via `https://youtu.be/JchvD4sMqBk`

Table 3.3 : Course topics of Lecture 3.

Lecture 3
3.1) Variables and tracking robot states
3.2) Usage of the status LEDs
3.3) Delayed or repetitive jobs
3.4) Usage of the accelerometer

3.3 Assessment Procedure

Each lesson has an exam which includes 10 objectives on the discussed topic on that day. Students are expected to solve and implement these objectives with Thymio robots using the respective interface. Each objective is designed independently and generally requires movement of the robot or generation of an output to a given input. For example, in first exam objectives, it's expected that the robot stops its movement and ends other outputs when it senses a tap on it. All three exam objectives regarding each lecture are given in Table 3.4, Table 3.5 and Table 3.6. After the exams, codes are collected for evaluation.

Table 3.4 : Exam 1 objectives.

Condition	**Expected Outcome**	**Related Topics**
Forward button is touched	Robot starts a forward movement at normal speed	1.1, 1.2, 1.3, 1.4, 1.6
Backward button is touched	Robot starts a backward movement at double speed	1.1, 1.2, 1.3, 1.4, 1.6
Center button is touched	Robot turns in clock-wise	1.1, 1.2, 1.3, 1.4, 1.6
Center button is touched	Robot turns on the top LEDs to red	1.1, 1.2, 1.3, 1.4, 1.7
Left button is touched	Robot starts to escape to left rear side	1.1, 1.2, 1.3, 1.4, 1.6
Right button is touched	Robot starts to escape to right front side	1.1, 1.2, 1.3, 1.4, 1.6
Right button is touched	Robot turns on the bottom LEDs to green	1.1, 1.2, 1.3, 1.4, 1.7
Robot senses a tap on it	Robot stops all movement immediately	1.1, 1.2, 1.3, 1.4, 1.6
Robot senses a tap on it	Robot turns off the top LEDs	1.1, 1.2, 1.3, 1.4, 1.7
Robot senses a tap on it	Robot turns off the bottom LEDs	1.1, 1.2, 1.3, 1.4, 1.7

Table 3.5 : Exam 2 objectives.

Condition	**Expected Outcome**	**Related Topics**
Forward button is touched	Robot starts a forward movement at normal speed	1.3, 1.4, 1.6
Center button is touched	Robot stops all movement immediately	1.3, 1.4, 1.6
An object is detected in front left side	Robot starts to escape to right rear side	1.3, 1.6, 2.1, 2.5
An object is detected in front right side	Robot starts to escape to left rear side	1.3, 1.6, 2.1, 2.5
An object in detected in front of the robot	Robot starts to escape to backward	1.3, 1.6, 2.1, 2.5
An object in detected in rear of the robot	Robot starts to escape to forward	1.3, 1.6, 2.2, 2.5
While robot is moving forward, it detects a cliff in front of it	Robot stops all movement immediately	1.3, 1.6, 2.3, 2.5
While robot is moving forward, it detects a cliff in front of it	Robot turns on the top LEDs to red	1.3, 1.7, 2.3, 2.5
Backward button is touched	Robot starts a backward movement at normal speed	1.3, 1.4, 1.6
Backward button is touched	Robot turns off the all top LEDs	1.3, 1.4, 1.7

At evaluation phase, collected codes were tested using the real robot in predesigned evaluation environment, which has obstacles and cliffs on necessary locations. Each objective was assessed in the given order and considered successful only if the code produces the expected outcome. Successful attempts were considered 1 point, while any other result is worth of 0 points. Thus, our evaluation focused on the outcome and became independent from the methods and algorithms or how the code is written. While interpreting the result of the experiment, we focused on two main criteria: success rate of the student in terms of exam objectives and time spent to finish the exam.

Table 3.6 : Exam 3 objectives.

Condition	**Expected Outcome**	**Related Topics**
While moving forward, robot senses a tap on it	Robot stops all movement immediately	1.3, 1.5, 2.4, 1.6, 3.1
While moving forward, robot senses a tap on it	Robot turns off the all top LEDs	1.3, 1.5, 1.7, 2.4, 3.1
While stopping, robot senses a tap on it	Robot starts a forward movement at normal speed	1.3, 1.5, 1.6, 2.4, 3.1
While stopping, robot senses a tap on it	Robot turns on the only front-left status LED	1.3, 1.5, 2.4, 3.1, 3.2
While moving forward, forward button is touched	Robot doubles its speed	1.3, 1.4, 1.6, 2.4, 3.1
While moving forward, robot senses a slope	Robot stops all movement immediately	1.3, 1.6, 2.4, 3.1, 3.4
While moving forward, robot senses a slope	Robot turns off the all top LEDs	1.3, 1.7, 2.4, 3.1, 3.4
Center button is touched	Robot turns on the all four status LEDs	1.3, 1.4, 3.2
Center button is touched	Robot delays the other actions 1 second	1.3, 1.4, 2.4, 3.1, 3.3
Center button is touched	Robot stops all the movement after a delay and disables itself	1.3, 1.4, 2.4, 3.1, 3.3

4. EXPERIMENTS AND RESULTS

Using Thymio and its programming interfaces, we gave programming education to students in three lectures. While teaching programming, we observed students' performance and compared the programming interfaces in terms of success rate and time. We gave the experimental results in the result section.

4.1 Experimental Conditions

We prepared two different interfaces: visual programming language VPL and text programming with AESL. For two interfaces, we prepared a common curriculum, same timetable and same exam questions, so that we can compare them within each group of students. In our experiments, 32 students, between ages of 7 to 24 with an average 14.3 and standard deviation 5.25 participated. Age distribution is shown in Fig. 4.1. In order to examine the effect of age, students were divided into two groups as younger and older than 14 years. The majority of the students had no experience with programming before and rest of them had vaguely information on programming. Their computer skills focused on the everyday usage and they had intermediate level of English proficiency.

Experiments were performed in an empty classroom with a maximum of two students. We provided a laptop, a computer mouse, a headphone, an educational robot and a large table for each student during the experiment. Students used the integrated laptop keyboard for coding in text programming and external mouse for visual interface. The headphones are used while they watching the video lectures on the computers in order to give them better hearing. Our sessions were held for maximum two students per day, thus they did not need to share any resources among the themselves.

Younger students were selected from The Educational Volunteers Foundation of Turkey (TEGV), which is a volunteer organization aims to support the primary education provided by the government. In the TEGV education plan, students participate in TEGV trainings which are scheduled periodically according to the their

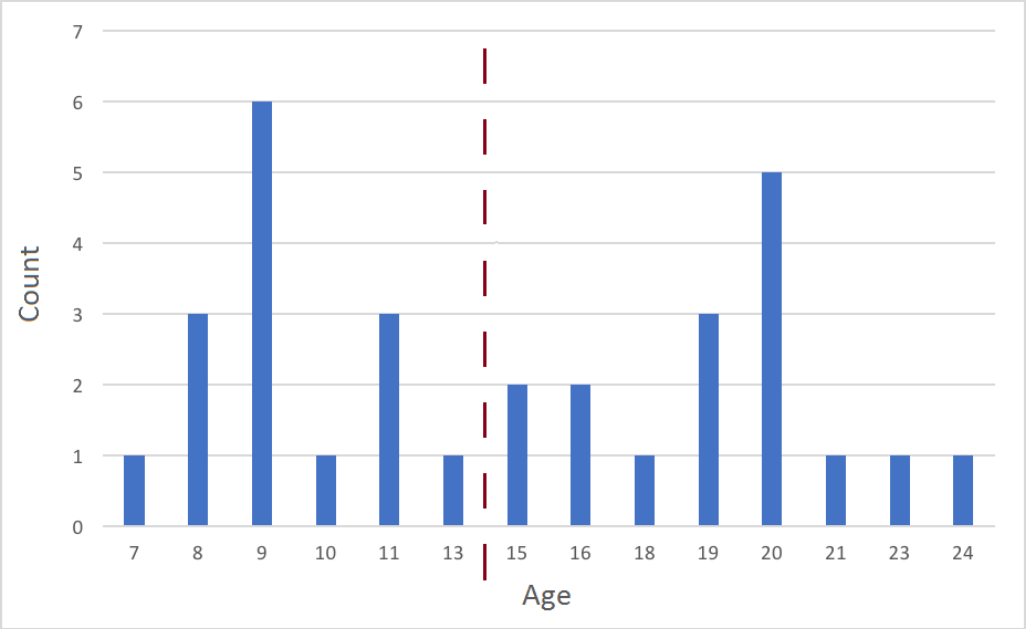

Figure 4.1 : Age distribution of students.

grades during the school term period and these trainings help to their education at the school. These trainings are usually held on saturday and sunday between 10:00 am and 4:00 pm, 6 hours a day. Usual lessons focused on the core subjects of the students' schools. However, TEGV gives effort to integrate an education plan which aims to provide students more practical lessons in order to they have fun while they learning. In this aspect, our robotic programming education classes fit their educational goals. So every weekend, we have worked with TEGV İpek Kıraç Unit and have taught two students each day.

Older students are selected from İstanbul High School and İstanbul Technical University. For high school students, our training schedule was planned according to their school timetables. On weekdays, we visited İstanbul High School after the regular classes were over and we gave maximum two hours of training a day. For the university students, we held our lessons in the classrooms of İstanbul Technical University. They have been selected from different departments which does not have any programming education and their lesson schedules were planned individually according the their timetables.

Class environments for students between ages of 7 - 13 are shown in Fig. 4.3 and 15-24 are shown in Fig. 4.2.

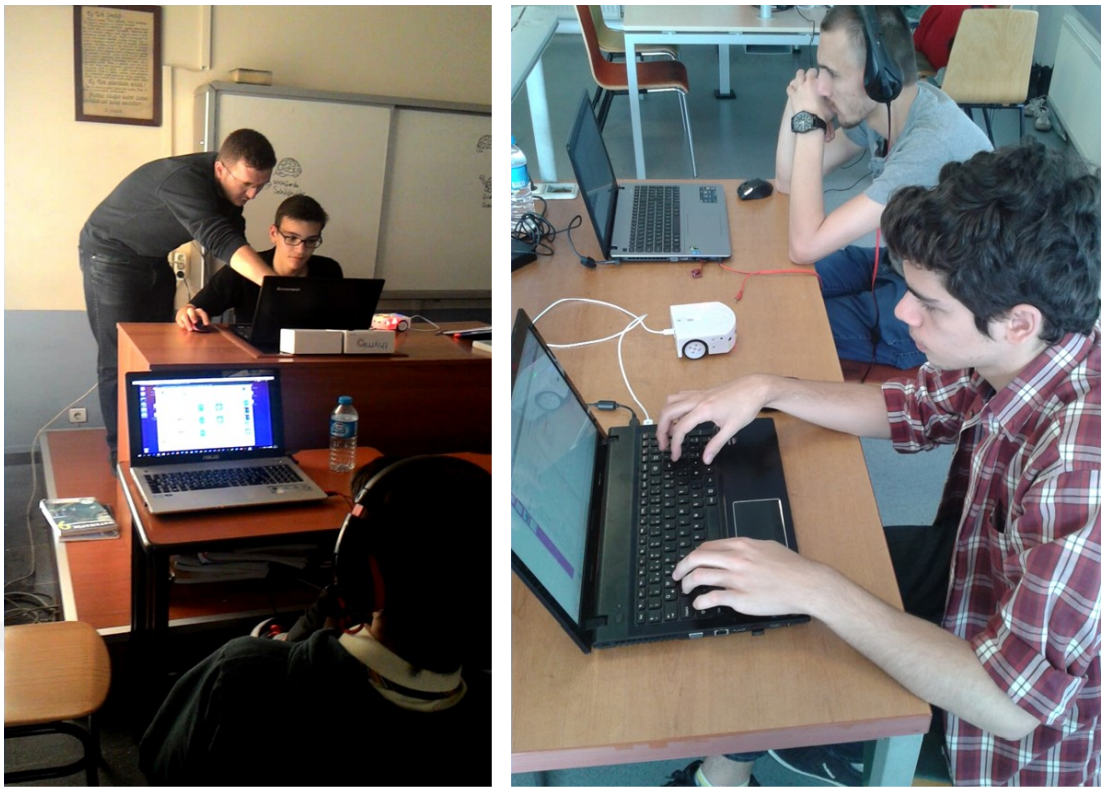

Figure 4.2 : Experiment environment: left) students in lecture, right) a student in test.

In order to compare the interfaces, we divided students into 3 different groups as below:

- Group 1 : Students who studied visual programming
- Group 2 : Students who studied text programming
- Group 3 : Students who studied text programming after visual programming

We distributed randomly selected students to the groups and gave efforts to keep average of the age evenly between groups. We taught visual programming to 20 students and only 12 of them continued to text programming for the Group 3. Hence, in order to keep numbers even, we taught 12 students text programming for Group 2.

We created Group 3 in order to evaluate the effect of visual languages on learning text-based languages. However, we did not create a group who studied visual programming after text programming, because our main objective was to teach students programming before moving to the advanced languages, which are all text-based. Also, in order to investigate the effects of the keywords and native function

Figure 4.3 : Class environment for younger students.

names of the programming languages, we divided our text programming students between English AESL and Turkish AESL.

4.2 Results

Test results of each group on each lesson and their averages are summarized in Table 4.1 with the completion time regarding each lesson in their respective interfaces. Maximum allowed time for each test is given as 15 minutes (900 seconds). The students generally did not use all the given time, so we could compared the completion time. But in third test of text programming interfaces, all students used the all given time to complete test. This can be explained with that the third test had much more complicated topics compared to first two test and text programming takes more time to complete these objectives. Also third test requires the knowledge of the first two lessons to complete given objectives and this increase the complexity of the tasks.

According to the results of experiment, students who studied with visual programming were more successful than students who studied only text programming. Group 1's success rate is distinctively higher than Group 2 in each lesson and this difference

Table 4.1 : Average results of the experiments.

Group	Attendees	Success Rate (out of 10)				Test Time (in seconds)			
		1st test	2nd test	3rd test	Average	1st test	2nd test	3rd test	Average
Group 1	20	9.55	9.35	9.10	9.33	348.15	369.80	681.60	466.52
Group 2	12	8.08	8.58	2.75	6.47	697.33	729.08	900.00	775.47
Group 3	12	9.33	9.25	5.33	7.97	524.42	729.92	900.00	718.11

is more apparent in the third lesson which has more complicated content compared to the first two lessons. In average, Group 1 is 44% better than Group 2 in terms of the success rates. For statistical significance, we applied Student's t-test to score data and we found that Group 1 and Group 2 are different with p-value 0.000 in 95% confidence. So, it can be said that the visual programming languages are easier for learning programming **(RQ1)**. Also, visual programming interfaces allow students to program faster than text based programming languages. Group 1's test durations are lower for each lesson than the ones of the Group 2. In average Group 2 takes 66% more time than Group 1 for the questions to be answered. Hence, it can be said that visual programming languages is also faster for learning programming **(RQ1)**.

When we compare students who studied only text programming (Group 2) with students who firstly studied visual programming then text programming (Group 3), we observed that the students in Group 3 have been 23% more successful than the ones in Group 2. For statistical significance, we applied Student's t-test to the score data and we found that Group 2 and Group 3 are different with p-value 0.001 in 95% confidence. This comparison show that learning programming with visual languages helps the students to learn text based programming languages later. When the complexity of the lessons and the tests increased, success rates became more distinct between Group 2 and Group 3. At the 3rd test, Group 3 doubled the score against Group 2. We also observed that while visual programming interfaces are faster and more effective in learning programming, they also increase the success rate of the text programming interfaces if they are used in advance in the curriculum **(RQ2)**.

Another measurement criterion on our test between Group 2 and Group 3 is the time spent on each interface. The effects of the visual programming languages also can be seen on the time spent for programming between students who studied visual programming first then text programming. Group 3 used 7% less time than Group

2. According to these results we confirm that learning programming with visual languages helps to learn text based programming faster **(RQ2)**.

According to age distribution which is shown in Fig. 4.1, we divided our students to two age groups as 7-13 and 15-24. Average results for each group are shown in Table 4.2 and Table 4.3. According to the results, older students had better success rates than younger students. In every test, younger students were below older students in terms of success rates. However, there was no significant difference between younger and older student groups in terms of test time. According to these results, age of the students has an effect on the success rate when learning programming **(RQ3)**, however we cannot draw a line for minimum age to learning programming because younger students are not far behind of older students. In addition to success rates, test times are decreased in Group 3 compared to Group 2 and this also compatible with the assumption.

Table 4.2 : Average results of the experiments for age group 7-13.

Group	Attendees	Success Rate (out of 10)				Test Time (in seconds)			
		1st test	2nd test	3rd test	Average	1st test	2nd test	3rd test	Average
Group 1	9	9.22	9.00	8.89	9.04	347.89	395.22	679.56	474.22
Group 2	6	7.67	8.17	2.67	6.17	699.67	727.83	900.00	775.83
Group 3	6	8.83	8.83	5.00	7.55	550.83	736.67	900.00	729.17

Table 4.3 : Average results of the experiments for age group 15-24.

Group	Attendees	Success Rate (out of 10)				Test Time (in seconds)			
		1st test	2nd test	3rd test	Average	1st test	2nd test	3rd test	Average
Group 1	11	9.82	9.64	9.27	9.58	348.36	349.00	683.27	460.21
Group 2	6	8.50	9.00	2.83	6.78	695.00	730.33	900.00	775.11
Group 3	6	9.83	9.67	5.67	8.39	498.00	723.17	900.00	707.06

Table 4.4 : Average results of Original AESL English and AESL Turkish.

Group	Attendees	Success Rate (out of 10)				Test Time (in seconds)			
		1st test	2nd test	3rd test	Average	1st test	2nd test	3rd test	Average
AESL Turkish	12	9.08	9.00	3.83	7.31	703.58	784.58	900.00	796.08
AESL English	12	8.33	8.83	4.25	7.14	518.08	674.12	900.00	697.50

We also took into account the effects of the names of keywords and native functions of the programming languages. Instead of English-based keywords and function names, we tested our subjects with a native language version AESL based on Turkish keywords and functions names. Results for these groups are shown in Table 4.4. However, in our test results, there was no significant difference between the success rates of original AESL and Turkish AESL. Average success rates were nearly identical between original AESL and Turkish AESL, so we did not include Turkish AESL on overall assessment as a different group. We conclude that the language of the keywords in text programming does not have any effect in learning programming **(RQ4)**.

5. CONCLUSION

In this study, we used an educational robot and different programming interfaces to teach programming to students between ages of 7 to 24 and observed the results of the teaching experiment. With using robots in our teaching method, we provide more tangible and active experience to students while learning basics of the programming. In their learning cycle, they learned the code, tried it on the robot and observed the output of their program physically. Thus, their attention focused on the robot and the program easily. We observed that they were very excited to use robots on the classroom and usually continued to study on robot even on their break time between lessons. Also they were eager to learn about new features of the robot and they spent time to try these features after the class is over. Moreover, after the tests, most of the students programmed the robot freely and they combined the different lecture contents to create new behaviours for the robot. After finishing the entire curriculum, some of the students came to class on different days to observe their peers and share their experience. This attitude confirms the positive effect of robots in programming education, which are supported with other studies [35] [37].

For the programming interfaces, we used both text programming interface and visual programming interface. We compared text-based programming interfaces against visual programming interfaces on learning programming first time. According to result of our experiments, visual programming languages are easier and quicker to learn. In the experiments, visual programming are higher success rate overall. When we compared the test completion times, visual programming interfaces are clearly took less time and it can be said that they are quicker in robot programming. In addition to that, when learning text programming, if students have visual programming background before, they have more successful results on learning text programming concepts. So, it can be said that if visual programming languages are used for learning programming in first time, students can adapt other programming languages easier. While previous works [27] suggest this idea according to pass rate of CS courses, we

provided detailed results for different tests and covered a wider age range. Also in our results difference between teaching approaches is more observable and statistically significant. We also concur with previous works [28] in that visual programming interfaces have positive influence on students when learning programming.

For the age groups, we divided our students into two age groups and investigated their success in learning programming. In contrary to the previous works [37] [27], we covered both younger and older students in same test environments and compared their performances. This comparison gave us a better understanding in observing the effect of the students' age in learning programming. According to results, older students have better average results in every test, so we can conclude that the age of the students an effect on the learning programming. However, difference between age groups is not distinct, so we can also conclude that while the age of students affects to the success rates, there was no breaking point in the results. Thus, there are no minimum age to teach programming. Also we compared the test completion time between two age groups and we observed that there is no difference between age groups and test completion times nearly identical. So, it can be said that the age of the students does not have any effect on learning time of programming.

For text programming interface, a Turkish keyword based programming language is also provided. Some of the students learned programming and took the test with this Turkish keyword based programming language. Results of the tests are identical to results of the original English keyword based programming language tests. So, according to our test, using a mother-tongue based programming languages does not have any effect on the learning programming. This can be explained with that either our students have proficient English knowledge or the text programming interfaces have a few keywords which can be remembered easily. If we took into account that both Turkish and English languages use same latin alphabet, students do not have any difficulties on remembering the keywords, even they do not need to know actual meaning of keywords.

In future works, other visual programming languages can be included in the experiments and can be compared with VPL. Also mixture of the text programming and visual programming interfaces can be different approach to teaching programming. So, these text and visual combined interfaces can be included the same tests. In

addition to that, teaching test can be compared with non robotic programming classes, so it can be shown how using robots in programming education affect the success rate of programming education.

REFERENCES

[1] **Feurzeig, W.** (1969). Programming-Languages as a Conceptual Framework for Teaching Mathematics: Final Report on the First Fifteen Months of the LOGO Project.

[2] **Clements, D.H. and Gullo, D.F.** (1984). Effects of computer programming on young children's cognition, *Journal of Educational Psychology*, *76*(6), 1051–1058.

[3] **Pea, R.D. and Kurland, D.M.** (1984). On the cognitive effects of learning computer programming, *New Ideas in Psychology*, *2*(2), 137–168.

[4] **Dalbey, J. and Linn, M.C.** (1985). The demands and requirements of computer programming: A literature review, *Journal of Educational Computing Research*, *1*(3), 253–274.

[5] **Saeli, M.**, **Perrenet, J.**, **Jochems, W.M.G. and Zwnaeveld, B.** (2010). Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective, *Informatics in Education*, *10*(1), 73–88.

[6] **Winslow, L.E.** (1996). Programming pedagogy - a psychological over-view, *ACM SIGCSE Bulletin*, *28*(3), 17–22.

[7] **Schoolnet, E.**, Computing our future Computer programming and coding - Priorities, school curricula and initiatives across Europe, `http://www.eun.org/c/document_library/get_file?uuid=521cb928-6ec4-4a86-b522-9d8fd5cf60ce&groupId=43887`, retrieved : 17.11.2017.

[8] **Pears, A.**, **Seidman, S.**, **Malmi, L.**, **Mannila, L.**, **Adams, E.**, **Bennedsen, J. and Paterson, J.** (2007). A survey of literature on the teaching of introductory programming, *ACM SIGCSE Bulletin*, *39*(4), 204.

[9] **Howe, E.**, **Thornton, M. and Weide, B.** (2004). Components-First Approaches to CS1/CS2: Principles and Practice.

[10] **Stein, L.A.**, (1998), What We've Swept Under the Rug: Radically Rethinking CS1.

[11] **Stein, L.A.**, (1999), Challenging the Computational Metaphor: Implications for How We Think, `http://cogprints.org/545/`.

[12] **Forte, A. and Guzdial, M.** (2005). Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses, *IEEE Transactions on Education*, *48*(2), 248–253.

[13] **Kay, D.G.** (1996). Bandwagons Considered Harmful, or the Past As Prologue in Curriculum Change, *SIGCSE Bull.*, *28*(4), 55–58, `http://doi.acm.org/10.1145/242649.242666`.

[14] **Eckerdal, A.**, **Thuné, M. and Berglund, A.** (2005). What Does It Take to Learn 'Programming Thinking'?, *Proceedings of the First International Workshop on Computing Education Research*, ICER '05, ACM, New York, NY, USA, pp.135–142.

[15] **Robins, A.**, **Rountree, J. and Rountree, N.** (2003). Learning and Teaching Programming: A Review and Discussion, *13*, 137–.

[16] **Trigwell, K.**, **Prosser, M. and Waterhouse, F.** (1999). Relations between teachers' approaches to teaching and students' approaches to learning, *Higher Education*, *37*(1), 57–70.

[17] **Cooper, S.**, **Dann, W. and Pausch, R.** (2003). Using Animated 3D Graphics to Prepare Novices for CS1, *13*, 3–30.

[18] **Jain, J.**, **Cross, II, J.H.**, **Hendrix, T.D. and Barowski, L.A.** (2006). Experimental Evaluation of Animated-verifying Object Viewers for Java, *Proceedings of the 2006 ACM Symposium on Software Visualization*, SoftVis '06, ACM, New York, NY, USA, pp.27–36.

[19] **Naps, T.L.** (2005). JHAVE: supporting algorithm visualization, *IEEE Computer Graphics and Applications*, *25*(5), 49–55.

[20] **Ala-Mutka, K.** (2005). A Survey of Automated Assessment Approaches for Programming Assignments, *15*, 83–102.

[21] **Douce, C.**, **Livingstone, D. and Orwell, J.** (2005). Automatic Test-based Assessment of Programming: A Review, *J. Educ. Resour. Comput.*, *5*(3).

[22] **Duke, R.**, **Salzman, E.**, **Burmeister, J.**, **Poon, J. and Murray, L.** (2000). Teaching programming to beginners – choosing the language is just the first step, *ACE 2000*, 79–86.

[23] **Kölling, M. and Rosenberg, J.** (1996). Blue - a language for teaching object-oriented programming, *ACM SIGCSE Bulletin*, *28*(1), 190–194.

[24] **Brilliant, S.S. and Wiseman, T.R.** (1996). The first programming paradigm and language dilemma, *ACM SIGCSE Bulletin*, *28*(1), 338–342.

[25] **Al-A'Ali, M. and Hamid, M.** (1995). Design of an arabic programming language (ARABLAN), *Computer Languages*, *21*(3-4), 191–201.

[26] **Maloney, J.**, **Resnick, M.**, **Rusk, N.**, **Silverman, B. and Eastmond, E.** (2010). The scratch programming language and environment, *ACM Transactions on Computing Education*, *10*(4), 1–15.

[27] **Rizvi, M. and Humphries, T.** (2012). A Scratch-based CS0 course for at-risk computer science majors, *2012 IEEE Frontiers in Education Conference (FIE)*, *00*, 1–5.

[28] **Malan, D.J. and Leitner, H.H.** (2007). Scratch for Budding Computer Scientists, *SIGCSE Bull.*, *39*(1), 223–227.

[29] **Peppler, K. and Kafai, Y.** (2009). Creative coding: Programming for personal expression.

[30] **Maloney, J.H.**, **Peppler, K.**, **Kafai, Y.**, **Resnick, M. and Rusk, N.** (2008). Programming by Choice: Urban Youth Learning Programming with Scratch, *SIGCSE Bull.*, *40*(1), 367–371.

[31] **Foundation, K.**, The Kojo Learning Environment, `http://www.kogics.net/kojo`, retrieved : 17.11.2017.

[32] **Regnell, J. and Pant, L.** (2014). Teaching programming to young learners using scala and kojo, *LTHs Pedagogiska Inspirationskonferens*, *8*, 4.

[33] **Lawhead, P.B.**, **Duncan, M.E.**, **Bland, C.G.**, **Goldweber, M.**, **Schep, M.**, **Barnes, D.J. and Hollingsworth, R.G.** (2002). A Road Map for Teaching Introductory Programming Using LEGO Mindstorms Robots, *SIGCSE Bull.*, *35*(2), 191–201.

[34] **Lindh, J. and Holgersson, T.** (2007). Does Lego Training Stimulate Pupils' Ability to Solve Logical Problems?, *Comput. Educ.*, *49*(4), 1097–1111.

[35] **Sullivan, F.R.** (2008). Robotics and science literacy: Thinking skills, science process skills and systems understanding, *Journal of Research in Science Teaching*, *45*(3), 373–394.

[36] **Doolittle, P. and Camp, W.** (1999). Constructivism: The Career and Technical Education Perspective, *J. Vocational and Technical Education*, *16*(1).

[37] **Bers, M.U.**, **Ponte, I.**, **Juelich, C.**, **Viera, A. and Schenker, J.** (2002). Teachers as designers: Integrating robotics in early childhood education, *Information Technology in Childhood Education Annual*, *2002*(1), 123–145.

[38] Home - Mindstorms LEGO.com, `https://www.lego.com/en-us/mindstorms`, retrieved : 17.11.2017.

[39] Arduino - Home, `https://www.arduino.cc`, retrieved : 17.11.2017.

[40] Overview & Download - Thymio & Aseba, `https://www.thymio.org/en:start`, retrieved : 17.11.2017.

[41] **Riedo, F.**, **Chevalier, M.**, **Magnenat, S. and Mondada, F.** (2013). Thymio II, a robot that grows wiser with children, *2013 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*.

[42] Specifications - Thymio & Aseba., `https://www.thymio.org/en:thymiospecifications`, retrieved : 17.11.2017.

[43] **Magnenat, S.**, **Rétornaz, P.**, **Bonani, M.**, **Longchamp, V. and Mondada, F.** (2011). ASEBA: A Modular Architecture for Event-Based Control of Complex Robots, *IEEE/ASME Transactions on Mechatronics*, *16*(2), 321–329.

[44] **Magnenat, S.**, **Rettornaz, P.**, **Bonani, M.**, **Longchamp, V. and F., M.** (2011). ASEBA: A modular architecture for event-based control of complex robots, *IEEE/ASME Transactions on Mechatronics*, *16*(2), 321–329.

[45] **Shin, J.**, **Siegwart, R. and Magnenat, S.** (2014). Visual Programming Language for Thymio II Robot, *Interaction Design and Children (IDC)*.

[46] **Magnenat, S.**, **Longchamp, V. and Mondada, F.** (2007). Aseba, an event-based middleware for distributed robot control, *Workshops and Tutorials CD IEEE-RSJ 2007 International Conference on Intelligent Robotsand Systems*.

CURRICULUM VITAE

Name Surname: Besim Baransel BAĞCI

Place and Date of Birth: Edirne 01.08.1991

E-Mail: baransel@itu.edu.tr

EDUCATION:

- **B.Sc.:** 2014, Istanbul Technical University - Computer Engineering

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- Bağcı B.B., Kamaşak M., Ince G. (2018) The Effect of the Programming Interfaces of Robots in Teaching Computer Languages. In: Lepuschitz W., Merdan M., Koppensteiner G., Balogh R., Obdržálek D. (eds) Robotics in Education. *RiE 2017. Advances in Intelligent Systems and Computing, vol 630. Springer, Cham*